

Multitape Alphabet Reduction

Formal Proof in Isabelle/HOL

András Z. Salamon Michael Wehar

August 26, 2026

Abstract

Machine-checked formal proof in Isabelle/HOL that an arbitrary finite tape alphabet can be reduced to a fixed four-element alphabet on a nondeterministic multitape Turing machine while preserving the accepted language and tape count — and determinism, in the deterministic special case — at the cost of a single constant-factor slowdown that grows only logarithmically with the alphabet size.

Given a well-formed multitape machine M whose tape alphabet Γ_M has at least four symbols, the operation $M'' = \text{alphabet_reduce } M$ produces a machine over the fixed four-element alphabet $\text{sym4} = \{\text{BLANK4}, \text{LE4}, \text{BIT0}, \text{BIT1}\}$ (a blank $\square$, a left-end marker $\triangleright$, and the two bits 0, 1). It simulates M under a fixed per-symbol binary encoding, spelling each Γ_M -symbol as a block of $\lceil \log_2 |\Gamma_M| \rceil$ bits, and it preserves determinism. This session (`Multitape_Alphabet_Reduction`) establishes:

- *Language preservation:* for every input word w , $\text{encode_input_ar } \Gamma_M \text{ bl}_M w \in L(M'') \iff w \in L(M)$ (theorem `alphabet_reduce_language`; the forward inclusion is also available separately as `alphabet_reduce_language_forward`).
- *Time bound:* if M accepts w within $T(|w|)$ steps, then M'' accepts $\text{encode_input_ar } \Gamma_M \text{ bl}_M w$ within $(6k + 1) b T(|w|)$ steps, where k is the number of tapes of M and $b = \lceil \log_2 |\Gamma_M| \rceil$ is the per-symbol block width (theorem `alphabet_reduce_time_explicit`). The slowdown is thus a single constant factor $(6k + 1) b$ multiplying the running time, with no additive or lower-order term; it is fixed by the machine — the tape count (linearly) and the alphabet size (logarithmically, through the block width b above) — and does not depend on the input. The classical existential form $dbT(|w|) + eb|w| + f$ is the corollary `alphabet_reduce_time`.
- *Well-formedness, determinism, and tape count:* M'' is a well-formed substrate machine (`alphabet_reduce_wf`) that preserves determinism (`alphabet_reduce_det`) and the tape count (`alphabet_reduce_preserves_tape_count`); its tape alphabet

is the full four-element type `sym4`, of cardinality exactly four (`alphabet_reduce_produces_alphabet_size_4`).

This is an elementary per-symbol multi-cell encoding leading to a reduction in the number of tape symbols: the folklore alphabet homomorphism onto a fixed small alphabet, stated (but not costed) by Balcázar, Díaz, and Gabarró [2, p. 23] and set within the wider machine-model-simulation landscape [7, 6].

The nondeterministic multitape substrate lives in the sibling `Multitape_TM_Substrate` session; this session depends on `Multitape_TM_Substrate` and `HOL-Library`.

Acknowledgement. This formalisation was developed with proof engineering by Anthropic’s Claude Opus 4.5, 4.6, 4.7, and 4.8 as the work progressed, together with other Claude family models in supporting roles. Claude Code was used to structure the work and for access to Isabelle/HOL. We also thank Arthur Freitas Ramos who provided feedback and also contributed several tooling improvements.

Contents

1	Overview	4
1.1	High-level overview	4
1.2	Glossary	5
1.3	Scope and limitations	7
1.4	Details of theorem statements	8
2	Alphabet reduction	11
2.1	Per-symbol encoder helpers	11
2.2	Per-symbol encoder	12
2.3	Per-symbol encoder lemmas	13
2.4	Per-symbol decoder helpers	16
2.5	Per-symbol decoder	16
2.6	Stage type and validity	19
2.7	Per-tape enumeration helpers	21
2.8	Per-substep dispatch helpers	22
2.9	Substep transition relations	24
2.10	Determinism preservation	34
2.11	Simulation correspondence	36
2.12	Per-substep simulation steps	40
2.13	Read phase	50
2.14	Write phase composition	71
2.15	Advance phase	92
2.16	Per- M -step simulation phase and the chunked engine	102
2.17	Top-level theorems	107

3	Alphabet reduction: reverse language inclusion	114
3.1	Terminal accept configuration	114
3.2	Source substep-tags partition the substep relations	115
3.3	Destination substep-tags carve the cycle's substep order	115
3.4	Compute-step inversion	116
3.5	Next-step inversion (cycle closure / halt dispatch)	117
3.6	Determinism of the read substep	118
3.7	Source disambiguation for the remaining substeps	119
3.8	Substep step semantics — <i>mttm_step</i> -level lands-at	120
3.9	Walker invariants — per-substep stage predicates	121
3.10	Step lifts — <i>mttm_step</i> to specific substep	123
3.11	Chain pinning in <i>mttm_step</i> (<i>ar_delta_read M</i>)	125
3.12	Read-phase leaves, sub-relation chain variants	127
3.13	Read-phase combiners, sub-relation chain variants	130
3.14	Write-phase leaves, sub-relation chain variants	136
3.15	Advance-phase leaves, sub-relation chain variants	139
3.16	Write-phase combiners, sub-relation chain variants	140
3.17	Walker preservation — per-substep M'-step lemmas	151
3.18	Cycle-close bridging lemma	152
3.19	Chunked reverse engine	153

1 Overview

1.1 High-level overview

In one sentence: this session proves that *any* multitape Turing machine can be rebuilt as an equivalent machine whose tape alphabet has only four symbols, at the cost of a constant-factor slowdown. The rebuilt machine keeps every property a later construction relies on: the accepted language, the number of tapes, structural well-formedness, and determinism.

The entire development concerns a single function, `alphabet_reduce`, which maps a machine M over an arbitrary finite tape alphabet to a machine $M'' = \text{alphabet_reduce } M$ over the fixed four-element alphabet `sym4`. Each headline theorem is a guarantee about this one function:

- the output really uses four symbols (`alphabet_reduce_produces_alphabet_size_4`);
- it accepts the same language, as an “if and only if” modulo a fixed input re-encoding (`alphabet_reduce_language`);
- it runs within a linear-time slowdown (`alphabet_reduce_time`);
- it is still a legal machine (`alphabet_reduce_wf`), with the same number of tapes (`alphabet_reduce_preserves_tape_count`) and the same determinism status (`alphabet_reduce_det`).

Two points help a first-time reader. First, a machine here may be non-deterministic by default: its transition is a *relation*, so acceptance means “some run reaches the accept state”, and these theorems are statements about nondeterministic machines. The deterministic case is singled out by a separate condition, `det_mttm`. Second, because the alphabet shrinks, the reduced machine cannot literally accept the same strings; instead each source symbol is re-spelled as a fixed block of `sym4` symbols by the encoding `encode_input_ar`, and “same language” is meant up to that translation.

Every theorem fixes an arbitrary M with no constraint on its behaviour, so each is a genuine “for all M ” result: read `alphabet_reduce M` as “apply this to *any* machine”.

The sibling entry `Multitape_Alphabet_Enlargement` goes the other way: it *enlarges* the alphabet to buy speed (the classical linear-speedup theorem).

Terminology and prior work. *Alphabet reduction* is our name for this construction, chosen to pair with the sibling *alphabet enlargement*; the classical literature does not usually present the two together. The reduction itself is a folklore normalisation: a multitape machine over an arbitrary finite tape alphabet Γ is simulated over a fixed small alphabet at a per-machine

constant-factor cost of $\lceil \log_2 |\Gamma| \rceil$ cells per symbol. Classic texts assume it rather than cost it. Balcázar, Díaz, and Gabarró [2, p. 23] state it directly — a machine, being a word over its alphabet, “can be translated into any other alphabet having at least two symbols via a homomorphism” (their example encodes into $\{0, 1\}$, en route to gödelizing Turing machines) — but do not analyse its time cost. Hopcroft and Ullman [5, §7.2] likewise *assume* each machine’s tape symbols are already binary-encoded (enough to justify a universal machine over a fixed alphabet) and leave the construction as a reader exercise with no time analysis. The restricted-machine survey of the era [4] catalogues the surrounding landscape without isolating it. This factor is a *fixed constant* once Γ is fixed, *not* a $\log T$ -style slowdown; that slowdown arises instead from the separate tape-count reduction, and only appears once the two are composed. Neither van Emde Boas’s simulation survey [7] — which frames such alphabet re-encodings under the *Invariance Thesis* — nor Papadimitriou’s textbook [6] isolates the reduction as a costed construction; both treat it as part of the general machine-model-simulation landscape.

Related AFP developments. The closest formalisation prior is inside Balbach’s `Cook_Levin` [1]: its `Two_Four_Symbols` theory re-encodes between a four-symbol content alphabet and two binary symbols by fixed two-cell pairs, an encoding utility whose time cost stays implicit in the surrounding framework. This entry generalises that picture — an arbitrary l -symbol alphabet reduced to a fixed four, with the constant-factor slowdown stated as an explicit top-level theorem — and, unlike Balbach’s deterministic development, proves the reduction for nondeterministic machines while separately preserving determinism. Xu, Zhang, Urban, and Joosten’s `Universal_Turing_Machine` [8] assumes a fixed binary alphabet outright. The substrate descends from Dalvit and Thiemann’s `Multitape_To_Singletape_TM` [3] but makes the tape count a value rather than a type parameter, so this reduction and its time bound hold, in a single theorem, for machines of any number of tapes.

1.2 Glossary

The machine type is `mttm`, the ten-component descriptor `MTTM  $Q \Sigma \Gamma bl le \delta s t r k$` : state set, input alphabet, tape alphabet, blank symbol, left endmarker, transition relation, start/accept/reject states, and tape count. Throughout, M is the source machine.

Projections. Positional selectors that read one component out of M :

- `k_tm` M — the tape count k_M (a natural number);
- `Sigma_tm` M — the input alphabet Σ_M ;

- $\Gamma_{\text{tm}} M$ — the tape alphabet Γ_M ;
- $\text{bl}_{\text{tm}} M$ — the blank symbol;
- $\text{delta}_{\text{tm}} M$ — the transition relation δ_M ;
- $\text{t}_{\text{tm}} M$ — the accept state.

Shared substrate predicates. Each entry opens with the gist, then the precise conditions.

valid_mttm M “ M is a legal machine”. The structural invariant: Q and Γ finite, $\Sigma \subseteq \Gamma$, the three distinguished states lie in Q , blank and endmarker are tape-but-not-input symbols, $\text{accept} \neq \text{reject}$, $k > 0$, the transition relation is correctly typed, the *left-endmarker read discipline* holds (reading the endmarker forces re-writing it and never moving off the left edge), and every tape beyond index k is frozen blank.

well_formed_mttm M a legal machine that also starts cleanly and keeps its endmarker. An abbreviation: **valid_mttm** M strengthened with *non-degeneracy* ($\text{start} \neq \text{accept}$, $\text{start} \neq \text{reject}$, $\text{endmarker} \neq \text{blank}$) and the endmarker *write discipline* **le_unique** M . Required wherever a run must genuinely start, the endmarker must be detectable, and it must never be forged.

le_unique M the left endmarker is never freshly planted. A machine property in the same family as **det_mttm** — an opt-in structural guarantee a downstream construction may rely on. Precisely: no transition writes the endmarker where it was not already reading it ($a'j = le \Rightarrow aj = le$ over δ). Folded into **well_formed_mttm**; the alphabet transformations use it to pin the endmarker, and the origin-floating speed-up wrapper is the one construction that deliberately forgoes it (so it is **valid_mttm** but not **well_formed_mttm**).

det_mttm M M is deterministic. The transition relation is a partial function: each (state, read-vector) pair has at most one successor triple — the functionality predicate layered on the otherwise relational δ .

$L(M) = \text{Lang_mttm } M$ the language M accepts. All input words w (with set $w \subseteq \Sigma_M$) from which some run reaches the accept state; acceptance is *existential* over runs — nondeterministic.

accepts_in_time_mttm M w t M accepts w within t steps. Weak time-bounded acceptance: some run of length at most t reaches the accept state.

The input lives on a work tape. The substrate has *no* separate read-only input tape: the input word is placed on tape 0, which is an ordinary work tape. This is why `valid_mttm` requires $\Sigma \subseteq \Gamma$ — every input symbol occupies a tape cell, so it must also be a tape symbol — and it is load-bearing for the alphabet transformations, which re-encode the input *in place* on tape 0 rather than copying it to a fresh tape.

Alphabet-reduction vocabulary.

$b = \max(1, \lceil \log_2 |\Gamma| \rceil)$ the *block width*: the number of sym4 cells that spell one source symbol, and the alphabet-dependent factor in the slowdown $(6k + 1)b$ (with $k = \mathbf{k_tm} M$ the tape count). For $|\Gamma| \geq 4$ the max is inactive, so $b = \lceil \log_2 |\Gamma| \rceil$; it is the Isabelle constant `block_width` Γ .

`encode_input_ar` Γ bl w the input encoding
`concat (map (encode_symbol  $\Gamma$   $bl$ )  $w$ )`: the flat concatenation of the per-symbol encodings. Each source symbol is spelled as a fixed-width block of b cells over $\{\mathbf{BIT0}, \mathbf{BIT1}\}$, the binary numeral of the symbol’s index under a *blank-anchored* enumeration of Γ (a bijection $\Gamma \rightarrow \{0, \dots, |\Gamma| - 1\}$ that sends the blank bl to 0). The markers `BLANK4` and `LE4` never occur inside an encoded symbol; blank-anchoring is what lets an all-`BLANK4` block decode back to bl .

`sym4` the four-element output alphabet $\{\mathbf{BLANK4}, \mathbf{LE4}, \mathbf{BIT0}, \mathbf{BIT1}\}$: the blank $\square$, the left endmarker $\triangleright$, and the two bits 0, 1.

`ar_stage` the per-state bookkeeping carried by M'' (substep tag, current tape, bit counter, symbol buffer, direction vector, position kind); this is why M'' has state type $'q \times 'a$ `ar_stage`.

1.3 Scope and limitations

A reader deciding whether this construction fits their needs should note four caveats.

- *Minimal alphabet.* The source must carry at least four tape symbols ($|\Gamma_M| \geq 4$). The block-encoding is injective for *any* alphabet (the width always fits $|\Gamma_M|$); the real requirement is `block_width` ≥ 2 (the walker’s block logic assumes at least two cells per symbol), for which $|\Gamma_M| \geq 4$ is the (loose) sufficient condition the theorems carry.
- *Same language, up to a fixed re-encoding.* Because the alphabet shrinks, M'' cannot literally accept M ’s strings; “same language” means M'' accepts `encode_input_ar` Γ_M bl_M w exactly when M accepts w , stated modulo that translation.

- *Constant-factor slowdown, not $\log T$.* The overhead is *fixed* once Γ_M is fixed: the exact factor is $(6k + 1)b$ for a k -tape machine, with $b = \lceil \log_2 |\Gamma_M| \rceil$ cells per symbol (so it grows only logarithmically with the alphabet size). This is *not* the $\log T$ -style slowdown of a tape-count reduction, which arises separately and only once the two are composed.
- *The slowdown jumps at powers of two.* As a function of the alphabet size the factor $\lceil \log_2 |\Gamma_M| \rceil$ is a *step function*, not a smooth one: it is constant on each band $2^{b-1} < |\Gamma_M| \leq 2^b$ and rises by one exactly as $|\Gamma_M|$ crosses a power of two. A machine over five symbols pays the same width ($b = 3$) as one over eight, but strictly more than one over four ($b = 2$); a symbol added within a band is free, one that crosses a power of two costs a whole extra bit. Both edges of the band are proved ($2^{b-1} < |\Gamma_M| \leq 2^b$), so b is exactly the width to index the whole alphabet, though one cell above the coding minimum $\lceil \log_2 (|\Gamma_M| - 1) \rceil$ for the non-blank symbols (index 0 is reserved for the blank, buying a branch-free decoder), the two agreeing except just past a power of two.

Each theorem’s exact hypotheses appear in the next subsection.

1.4 Details of theorem statements

For each headline theorem we give a cleaned statement and the role of every predicate it mentions. The hypotheses form a *ladder*: purely structural facts need none, validity needs the minimal-alphabet bound $|\Gamma_M| \geq 4$, behavioural equivalence needs the stronger `well_formed_mttm`, and determinism preservation needs `valid_mttm` together with `det_mttm`.

`alphabet_reduce_produces_alphabet_size_4.`

$$|\text{sym4}| = 4.$$

No machine and no hypotheses. This is the operational meaning of “reduction to four”: the reduction hard-codes the output tape alphabet as the whole type `sym4`, and this theorem certifies that type has exactly four inhabitants. It is a fact about the target *type*, deliberately independent of any particular M .

`alphabet_reduce_preserves_tape_count.`

$$k_{M''} = k_M.$$

Mentions only the tape-count projection `k_tm` and the reduction, with no hypotheses. The reduction acts on the alphabet dimension alone; the tape count is copied verbatim, so the equation holds for every term of machine type, valid or not.

alphabet_reduce_wf.

$$\text{valid_mttm } M \wedge |\Gamma_M| \geq 4 \implies \text{valid_mttm } M''.$$

Closure of the structural invariant; **valid_mttm** appears as both hypothesis and conclusion. The side condition $|\Gamma_M| \geq 4$ is the *minimal-alphabet* requirement: the source must carry at least four tape symbols for the binary block-encoding to be well-defined and injective. Discharging it amounts to re-checking the endmarker discipline and frozen-tape clauses in the encoded world.

alphabet_reduce_language.

$$\begin{aligned} &\text{well_formed_mttm } M \wedge |\Gamma_M| \geq 4 \implies \\ &\forall w. \text{ set } w \subseteq \Sigma_M \longrightarrow \\ &\quad (\text{encode_input_ar } \Gamma_M \text{ bl}_M w \in L(M'')) \\ &\quad = (w \in L(M)). \end{aligned}$$

The correctness core, stated as an equivalence between two membership facts. Note the upgrade from **valid_mttm** to **well_formed_mttm**: the simulation needs the start state to be a genuine pre-halt state and the endmarker to be distinguishable from the blank. The guard $\text{set } w \subseteq \Sigma_M$ restricts attention to genuine input words, and **encode_input_ar** is the translation under which the two languages coincide. The forward inclusion ($\Rightarrow$) is available separately as **alphabet_reduce_language_forward**; this theorem strengthens it to the equivalence.

alphabet_reduce_time_explicit and alphabet_reduce_time. Write k for the number of tapes of M and $b = \text{block_width}(\Gamma_M) = \lceil \log_2 |\Gamma_M| \rceil$ for the per-symbol block width. Under $\text{well_formed_mttm } M \wedge |\Gamma_M| \geq 4$,

$$\begin{aligned} &\forall w. \text{ set } w \subseteq \Sigma_M \longrightarrow \\ &\quad \text{accepts_in_time_mttm } M w (T(|w|)) \longrightarrow \\ &\quad \text{accepts_in_time_mttm } M'' \\ &\quad (\text{encode_input_ar } \Gamma_M \text{ bl}_M w) ((6k + 1) b T(|w|)). \end{aligned}$$

Linear-time slowdown with an *explicit* constant. The factor is the single number $(6k + 1) b$ multiplying the running time, with no additive and no linear-in- $|w|$ term. It comes from an exact per-step super-step count of $k(5b + 2) + 2$, folded up to $(6k + 1) b$ using $b \geq 2$ (which holds because $|\Gamma_M| \geq 4$) and tight at $b = 2$, i.e. $|\Gamma_M| = 4$. The crux is that this constant is bound *once, outside the* $\forall w$: it depends only on M (its tape count, and logarithmically its alphabet size), so the slowdown is uniform across all inputs rather than tuned per input. The classical existential form $\exists d, e, f, b \geq$

1. $dbT(|w|) + eb|w| + f$ is the obtains-corollary `alphabet_reduce_time`, hiding the explicit constants behind existentials instantiated at $d = 6k + 1$ and $e = f = 0$.

`alphabet_reduce_det.`

$$\text{valid_mttm } M \wedge \text{det_mttm } M \implies \text{det_mttm } M''.$$

Determinism preservation. It needs *both* `valid_mttm` and `det_mttm`: functionality of the reduced transition relation follows from functionality of the source only because validity pins the transition typing on which the per-substep transitions of M'' are built. The proof factors through one functionality lemma per substep (read, compute, write, advance, next).

```

theory AlphabetReduction_Codec
  imports Multitape_TM_Substrate.Multitape_Substrate
begin

```

2 Alphabet reduction

This theory opens the alphabet-reduction development. The *alphabet_reduce* combinator takes a well-formed substrate machine over an arbitrary finite alphabet *a* (with tape-alphabet cardinality ≥ 4) and produces a well-formed substrate machine over the fixed four-element alphabet *sym4*, preserving the accepted language, determinism, and tape count.

This is the elementary determinism-preserving per-symbol multi-cell encoding.

The development is layered: this first theory fixes the output alphabet *sym4* and the per-symbol binary codec; the simulator, its delta, and the headline theorems (*alphabet_reduce_wf*, *alphabet_reduce_language*, *alphabet_reduce_time*, *alphabet_reduce_det*) are built across the theories that follow. Throughout, *b* abbreviates *block_width* Γ (the per-symbol cell width).

The fixed four-element output alphabet: *BLANK4* (blank), *LE4* (left endmarker), *BIT0* (binary 0), *BIT1* (binary 1).

```

datatype sym4 = BLANK4 | LE4 | BIT0 | BIT1

```

```

lemma sym4_UNIV: (UNIV :: sym4 set) = {BLANK4, LE4, BIT0, BIT1}
  <proof>

```

```

lemma sym4_card: card (UNIV :: sym4 set) = 4
  <proof>

```

2.1 Per-symbol encoder helpers

The per-source-symbol block length: the number of *sym4* cells that encode one Γ -symbol, and the slowdown factor of the reduction. At least 1 (to avoid degenerate empty encodings even for trivial Γ), and otherwise $\lceil \log_2 (\text{card } \Gamma) \rceil$, the fewest binary digits that index all of Γ . Defined combinatorially via *LEAST* on the predicate $\text{card } \Gamma \leq 2 \wedge n$; existence of such an *n* follows from $\text{card } \Gamma \leq 2 \wedge \text{card } \Gamma$ (HOL.Power.less_exp).

What is counted. The index space $\{.. < \text{card } \Gamma\}$ covers *every* symbol of Γ : the blank at index 0 (the anchor *gamma_enum* relies on — see below), and the left endmarker *le* as an ordinary coded symbol, since *valid_mttm* permits a machine to write *le* at any tape position (its only special handling is the single *LE4* cell at position 0). So *block_width* is genuinely the width of this uniform code, not a loose bound on a smaller one — but it is not the *coding* minimum. Reserving index 0 for the blank (so an all-*BLANK4* block reads back through the same accumulator) puts it one digit above

$\lceil \log_2 (\text{card } \Gamma - 1) \rceil$, the width for the $\text{card } \Gamma - 1$ non-blank symbols; the two agree except just past a power of two. Detecting the all-*BLANK4* block directly would recover that digit, and additionally giving *le* its own marker block would reach $\lceil \log_2 (\text{card } \Gamma - 2) \rceil$ — but each costs a marker-detection branch in the read and write phases, so that constant-factor gain is deliberately not taken.

definition *block_width* :: 'a set $\Rightarrow$ nat **where**
block_width $\Gamma = \max 1 \ (\text{LEAST } n. \text{card } \Gamma \leq 2 \wedge n)$

A *blank-anchored* enumeration of Γ as a bijection to $\{..< \text{card } \Gamma\}$ that additionally sends the blank *bl* to 0. The anchor is load-bearing for the read phase: a blank source cell is encoded as a *b*-cell all-*BLANK4* block (*bit_value* = 0), so the read fold is a fixpoint at *gamma_unenum* Γ *bl* 0, which must equal *bl* — and the decode round-trips force that to *gamma_enum* Γ *bl* *bl* = 0. Existence of such a bijection is unconditional on *finite* Γ for *any* *bl* (*ex_bij_betw_anchor* below: swap 0 with *g bl* when *bl* $\in \Gamma$, otherwise free off-domain), so the carried *bl* needs no *bl* $\in \Gamma$ side condition. Downstream proofs reason about *bij_betw*- and anchor-derived facts only; the Hilbert choice picks one specific anchored bijection.

definition *gamma_enum* :: 'a set $\Rightarrow$ 'a $\Rightarrow$ 'a $\Rightarrow$ nat **where**
gamma_enum Γ *bl* =
 (SOME *f*. *bij_betw* *f* Γ $\{..< \text{card } \Gamma\} \wedge f \text{ bl} = 0$)

Render a natural number as a fixed-length *sym4* bit list (*BIT0* / *BIT1* only). The list has shape $[bit_{b-1}, \dots, bit_1, bit_0]$: most-significant bit at the head, least-significant bit at the tail. For $n \geq 2 \wedge b$ the high bits are dropped (only the low *b* bits are rendered).

fun *nat_to_bits* :: nat $\Rightarrow$ nat $\Rightarrow$ *sym4* list **where**
nat_to_bits 0 _ = []
 | *nat_to_bits* (Suc *k*) *n* =
 nat_to_bits *k* (*n* div 2)
 @ [if *n* mod 2 = 0 then *BIT0* else *BIT1*]

2.2 Per-symbol encoder

Encoding a single 'a-symbol as a fixed-length list of *sym4*-symbols, parameterised over the source alphabet Γ (a finite set, passed explicitly as a value rather than through a type-class constraint, so the construction is uniform over all source alphabets). The length *b* is the slowdown factor; the encoding is injective on Γ . The two bit symbols *BIT0*, *BIT1* carry the payload; *LE4* and *BLANK4* do not appear in any encoder image. For $x \notin \Gamma$ the encoder produces a value that's structurally well-shaped (a *b*-cell bit list) but semantically arbitrary — downstream proofs always operate on $x \in \Gamma$.

definition *encode_symbol* ::
 'a set $\Rightarrow$ 'a $\Rightarrow$ 'a $\Rightarrow$ *sym4* list **where**

$$\text{encode_symbol } \Gamma \text{ } bl \text{ } x = \text{nat_to_bits } (\text{block_width } \Gamma) (\text{gamma_enum } \Gamma \text{ } bl \text{ } x)$$

Encoding an input word as a flat list of *sym4*-symbols (concatenation of per-symbol encodings under the given Γ).

definition *encode_input_ar* ::

'a set $\Rightarrow$ *'a* $\Rightarrow$ *'a list* $\Rightarrow$ *sym4 list* **where**

encode_input_ar $\Gamma \text{ } bl \text{ } w = \text{concat } (\text{map } (\text{encode_symbol } \Gamma \text{ } bl) \text{ } w)$

2.3 Per-symbol encoder lemmas

The encoded list has length b , by induction on the block width. Total — no $x \in \Gamma$ precondition is needed.

lemma *length_nat_to_bits* [*simp*]:

length (*nat_to_bits* $k \text{ } n$) = k

<proof>

lemma *length_encode_symbol* [*simp*]:

length (*encode_symbol* $\Gamma \text{ } bl \text{ } x$) = *block_width* Γ

<proof>

Every cell of an encoded symbol is in the bit alphabet $\{BIT0, BIT1\}$. In particular, the substrate-reserved markers *LE4* and *BLANK4* never appear in an encoder image.

lemma *set_nat_to_bits*:

set (*nat_to_bits* $k \text{ } n$) $\subseteq \{BIT0, BIT1\}$

<proof>

lemma *encode_symbol_cell_domain*:

set (*encode_symbol* $\Gamma \text{ } bl \text{ } x$) $\subseteq \{BIT0, BIT1\}$

<proof>

The enumeration *gamma_enum* $\Gamma \text{ } bl$ is a genuine bijection from Γ to $\{.. $\text{card } \Gamma$ \}$ whenever Γ is finite. Existence is library (*ex_bij_betw_finite_nat*); the Hilbert-choice operator picks one such bijection.

Existence of a blank-anchored bijection $\Gamma \rightarrow \{.. $\text{card } \Gamma$ \}$ sending *bl* to 0, unconditional on *finite* Γ for any *bl*. Take any bijection *g* (library *ex_bij_betw_finite_nat*); if *bl* $\in \Gamma$, post-compose with the nat-level transposition of 0 and *g bl* (both $< \text{card } \Gamma$), which is a *bij_betw* of $\{.. $\text{card } \Gamma$ \}$ onto itself by *endo_inj_surj*; if *bl* $\notin \Gamma$, just override *g* at *bl* (off-domain, so *bij_betw* is unchanged by *bij_betw_cong*).

lemma *ex_bij_betw_anchor*:

fixes $\Gamma :: 'a \text{ set}$ **and** $bl :: 'a$

assumes *finG*: *finite* Γ

shows $\exists f. \text{bij_betw } f \text{ } \Gamma \{.. $\text{card } \Gamma$ \} \wedge f \text{ } bl = 0$

<proof>

lemma *gamma_enum_anchor*:

assumes *finite* Γ
shows *bij_betw* (*gamma_enum* Γ *bl*) $\Gamma \{..< \text{card } \Gamma\}$
 $\wedge \text{gamma_enum } \Gamma \text{ bl bl} = 0$
 $\langle \text{proof} \rangle$

lemma *gamma_enum_bij*:
assumes *finite* Γ
shows *bij_betw* (*gamma_enum* Γ *bl*) $\Gamma \{..< \text{card } \Gamma\}$
 $\langle \text{proof} \rangle$

The anchor: the blank-anchored enumeration sends the blank to 0. This is what makes an all-*BLANK* block decode to *bl* (*decode_blank*).

lemma *gamma_enum_blank*:
assumes *finite* Γ
shows *gamma_enum* Γ *bl* *bl* = 0
 $\langle \text{proof} \rangle$

lemma *gamma_enum_lt_card*:
assumes *finite* Γ **and** $x \in \Gamma$
shows *gamma_enum* Γ *bl* $x < \text{card } \Gamma$
 $\langle \text{proof} \rangle$

The cardinality of Γ fits in b bits: $\text{card } \Gamma \leq 2^b$. This is the structural bound that underwrites injectivity: every enumeration value *gamma_enum* Γ *bl* $x < \text{card } \Gamma$ for $x \in \Gamma$ is in the range where *nat_to_bits* is injective.

lemma *card_le_two_pow_block_width*: $\text{card } \Gamma \leq 2^{\text{block_width } \Gamma}$
 $\langle \text{proof} \rangle$

Minimality (the lower edge): b is the *smallest* width that indexes all of Γ , so once $\text{card } \Gamma \geq 2$ one bit fewer does not suffice — $2^{b-1} < \text{card } \Gamma$. With *card_le_two_pow_block_width* this pins b to $\lceil \log_2 (\text{card } \Gamma) \rceil$ exactly: b is a *step function* of $\text{card } \Gamma$, constant on each band $2^{b-1} < \text{card } \Gamma \leq 2^b$ and jumping by one as $\text{card } \Gamma$ crosses a power of two ($4 \rightarrow 5$, $8 \rightarrow 9$, $16 \rightarrow 17$). The slowdown factor therefore rises in unit steps at the powers of two, not smoothly with alphabet size. This is minimality for the *uniform* code that indexes every symbol of Γ with the blank at index 0; the *coding* minimum (the non-blank symbols alone) sits one digit lower just past each power of two — see *block_width*.

lemma *two_pow_block_width_pred_less_card*:
assumes *card2*: $2 \leq \text{card } \Gamma$
shows $2^{(\text{block_width } \Gamma - 1)} < \text{card } \Gamma$
 $\langle \text{proof} \rangle$

The bit-renderer is injective on inputs bounded by 2^b : two values with the same b -bit representation are equal. Proven by induction on b : the head of the bit list determines the high bit $n \text{ div } 2$ (recursive case), and the tail single element determines the low bit $n \text{ mod } 2$; combining via $n = 2 \cdot (n \text{ div } 2) + n \text{ mod } 2$ gives equality.

lemma *nat_to_bits_inj_bounded*:
assumes $n1 < 2^k$
and $n2 < 2^k$
and $\text{nat_to_bits } k \ n1 = \text{nat_to_bits } k \ n2$
shows $n1 = n2$
 $\langle \text{proof} \rangle$

Injectivity-on- Γ : distinct source symbols in Γ produce distinct encoder images. This is the load-bearing property for language preservation: no two source symbols collide under the encoder.

lemma *encode_symbol_inj_on_Gamma*:
assumes *finite* Γ
and $x \in \Gamma$ **and** $y \in \Gamma$
and $\text{encode_symbol } \Gamma \ bl \ x = \text{encode_symbol } \Gamma \ bl \ y$
shows $x = y$
 $\langle \text{proof} \rangle$

Structural properties of the input-word encoder. These follow directly from the *concat* (*map* (*encode_symbol* $\Gamma \ bl$) ...) definition and are useful for the validation-phase reach lemmas in later slices.

lemma *encode_input_ar_Nil* [*simp*]:
 $\text{encode_input_ar } \Gamma \ bl \ [] = []$
 $\langle \text{proof} \rangle$

lemma *encode_input_ar_append*:
 $\text{encode_input_ar } \Gamma \ bl \ (w1 @ w2)$
 $= \text{encode_input_ar } \Gamma \ bl \ w1 @ \text{encode_input_ar } \Gamma \ bl \ w2$
 $\langle \text{proof} \rangle$

lemma *length_encode_input_ar*:
 $\text{length } (\text{encode_input_ar } \Gamma \ bl \ w) = \text{block_width } \Gamma * \text{length } w$
 $\langle \text{proof} \rangle$

Uniform-width block indexing: when every block $f \ x$ has the same length K , the $(i \cdot K + j)$ -th cell of *concat* (*map* $f \ xs$) is the j -th cell of the i -th block. The list-level fact underlying the initial-tape correspondence: *encode_input_ar* is exactly such a uniform concatenation, every block b wide by *length_encode_symbol*.

lemma *nth_concat_map_uniform*:
assumes $K: \bigwedge x. \text{length } (f \ x) = K$
and $i: i < \text{length } xs$
and $j: j < K$
shows $\text{concat } (\text{map } f \ xs) ! (i * K + j) = f \ (xs ! i) ! j$
 $\langle \text{proof} \rangle$

lemma *nth_encode_input_ar*:
assumes $i < \text{length } w$ **and** $j < \text{block_width } \Gamma$
shows $\text{encode_input_ar } \Gamma \ bl \ w ! (i * \text{block_width } \Gamma + j)$

$= \text{encode_symbol } \Gamma \text{ bl } (w ! i) ! j$

$\langle \text{proof} \rangle$

2.4 Per-symbol decoder helpers

Bit value of a *sym4* cell. *BIT1* contributes 1, every other symbol contributes 0; the non-bit symbols (*BIT0*, *LE4*, *BLANK4*) are conflated to zero — junk-in, junk-out — so the decoder produces a well-defined nat even on malformed input. Validity checking is performed separately by *decode_symbol*'s cell-domain test.

```
fun bit_value :: sym4  $\Rightarrow$  nat where
  bit_value BIT0 = 0
| bit_value BIT1 = 1
| bit_value LE4 = 0
| bit_value BLANK4 = 0
```

Read a *sym4* list as a binary number, MSB at the head (matches *nat_to_bits*'s output shape). Each position contributes *bit_value* times the appropriate power of two.

```
fun bits_to_nat :: sym4 list  $\Rightarrow$  nat where
  bits_to_nat [] = 0
| bits_to_nat (b # bs) = bit_value b * 2 ^ length bs + bits_to_nat bs
```

Snoc-form of *bits_to_nat*: appending a low bit shifts the existing value left and adds the new bit. This is the inductive workhorse for the round-trip lemma — it matches *nat_to_bits*'s recursive shape (which appends the LSB at the tail).

```
lemma bits_to_nat_snoc:
  bits_to_nat (xs @ [b]) = 2 * bits_to_nat xs + bit_value b
 $\langle \text{proof} \rangle$ 
```

Round-trip on bounded naturals: rendering *n* as a *b*-bit list and reading it back recovers *n*, provided $n < 2^b$. Proof by induction on *b*: the snoc-form of *bits_to_nat* peels off the trailing bit, the IH handles the prefix $n \text{ div } 2$, and $n = 2 \cdot (n \text{ div } 2) + n \text{ mod } 2$ reassembles.

```
lemma bit_value_low_bit:
  bit_value (if n mod 2 = 0 then BIT0 else BIT1) = n mod 2
 $\langle \text{proof} \rangle$ 
```

```
lemma bits_to_nat_nat_to_bits:
  assumes  $n < 2^k$ 
  shows bits_to_nat (nat_to_bits k n) = n
 $\langle \text{proof} \rangle$ 
```

2.5 Per-symbol decoder

Partial inverse of *encode_symbol*. Returns *Some x* when the input list:

1. has length b (correct cell count);
2. contains only *BIT0* / *BIT1* cells (no substrate-reserved markers); and
3. has binary interpretation strictly below $\text{card } \Gamma$ (in the enumeration range).

Otherwise returns *None*. The validation phase in later slices implements this check as a sequence of substep transitions; the abstract *decode_symbol* partial function is the specification target.

definition *decode_symbol* ::
 $'a \text{ set} \Rightarrow 'a \Rightarrow \text{sym4 list} \Rightarrow 'a \text{ option}$ **where**
decode_symbol $\Gamma \text{ bl } \text{ys} =$
 (if $\text{length } \text{ys} = \text{block_width } \Gamma$
 $\wedge \text{set } \text{ys} \subseteq \{\text{BIT0}, \text{BIT1}\}$
 $\wedge \text{bits_to_nat } \text{ys} < \text{card } \Gamma$
 then $\text{Some } (\text{inv_into } \Gamma (\text{gamma_enum } \Gamma \text{ bl}) (\text{bits_to_nat } \text{ys}))$
 else *None*)

Round-trip: decoding an encoded symbol from Γ recovers the source value. All three validity preconditions of *decode_symbol* are satisfied by the encoder image; *inv_into* resolves to x via *gamma_enum*'s injectivity-on- Γ .

lemma *decode_symbol_encode_symbol*:
assumes *finite* Γ **and** $x \in \Gamma$
shows *decode_symbol* $\Gamma \text{ bl } (\text{encode_symbol } \Gamma \text{ bl } x) = \text{Some } x$
 (proof)

A total form of the symbol decoder, indexed by a natural number rather than a bit list. For $n < \text{card } \Gamma$, returns the unique $x \in \Gamma$ with *gamma_enum* $\Gamma \text{ bl } x = n$ (via *inv_into*); for $n \geq \text{card } \Gamma$, returns the blank *bl* as a defensive fallback. Used by *ar_delta_read*'s per-bit accumulator to maintain *buf* as a partial-decoded *'a* value: at every per-bit substep before the boundary, the partial nat is strictly less than $2^b - 1 < \text{card } \Gamma$ (since b is the *least* n with $\text{card } \Gamma \leq 2^n$), so the fallback is never substantively reached for valid inputs; only the boundary substep can land out of range under non-encoder-image inputs, where the language theorem doesn't care.

definition *gamma_unenum* :: $'a \text{ set} \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a$ **where**
gamma_unenum $\Gamma \text{ bl } n =$
 (if $n < \text{card } \Gamma$
 then $\text{inv_into } \Gamma (\text{gamma_enum } \Gamma \text{ bl}) n$
 else *bl*)

Enumeration round-trips between *gamma_enum* and its total inverse *gamma_unenum*, the arithmetic core the read phase's incremental decode rests on. *gamma_enum* $\circ$ *gamma_unenum* is the identity on the in-range index set $\{0..<\text{card } \Gamma\}$ (where *gamma_unenum* resolves to *inv_into* and *gamma_enum* is surjective onto that set), and *gamma_unenum* $\circ$

gamma_enum is the identity on Γ (where gamma_enum is injective and lands below $\text{card } \Gamma$).

lemma $\text{gamma_enum_gamma_unenum}$:
assumes $\text{finite } \Gamma$ **and** $n < \text{card } \Gamma$
shows $\text{gamma_enum } \Gamma \text{ bl } (\text{gamma_unenum } \Gamma \text{ bl } n) = n$
 $\langle \text{proof} \rangle$

lemma $\text{gamma_unenum_gamma_enum}$:
assumes $\text{finite } \Gamma$ **and** $x \in \Gamma$
shows $\text{gamma_unenum } \Gamma \text{ bl } (\text{gamma_enum } \Gamma \text{ bl } x) = x$
 $\langle \text{proof} \rangle$

Reading an encoded symbol back via bits_to_nat recovers its enumeration index, the named form of the local bn fact inside $\text{decode_symbol_encode_symbol}$. Needed by the read-phase accumulator below.

lemma $\text{bits_to_nat_encode_symbol}$:
assumes $\text{finite } \Gamma$ **and** $x \in \Gamma$
shows $\text{bits_to_nat } (\text{encode_symbol } \Gamma \text{ bl } x) = \text{gamma_enum } \Gamma \text{ bl } x$
 $\langle \text{proof} \rangle$

The read-phase accumulator: one bit-cell folded into the running symbol. Mirrors the per-bit update the read substeps perform — $\text{buf} := \text{gamma_unenum } (2 \cdot \text{gamma_enum } \text{buf} + \text{bit_value } c)$ — reading the b -cell block most-significant bit first.

definition $\text{ar_acc} :: 'a \text{ set} \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{sym4} \Rightarrow 'a$ **where**
 $\text{ar_acc } \Gamma \text{ bl } b \text{ c} = \text{gamma_unenum } \Gamma \text{ bl } (2 * \text{gamma_enum } \Gamma \text{ bl } b + \text{bit_value } c)$

Folding the accumulator from the seed $\text{gamma_unenum } 0$ over a bit list whose value is in enumeration range yields exactly $\text{gamma_unenum } (\text{bits_to_nat } \text{ys})$. Proof by rev_induct : appending a low bit shifts the running value left and adds it (bits_to_nat_snoc), the prefix value stays in range ($\text{bits_to_nat } \text{zs} \leq \text{bits_to_nat } (\text{zs} @ [b])$, no separate take -bound lemma needed), and the $\text{gamma_enum} \circ \text{gamma_unenum}$ round-trip cancels at each step.

lemma foldl_ar_acc_eq :
assumes $\text{finite } \Gamma$
shows $\text{set } \text{ys} \subseteq \{\text{BIT0}, \text{BIT1}\} \implies \text{bits_to_nat } \text{ys} < \text{card } \Gamma$
 $\implies \text{foldl } (\text{ar_acc } \Gamma \text{ bl}) (\text{gamma_unenum } \Gamma \text{ bl } 0) \text{ ys}$
 $= \text{gamma_unenum } \Gamma \text{ bl } (\text{bits_to_nat } \text{ys})$
 $\langle \text{proof} \rangle$

Read-phase decode correctness: folding the accumulator over a symbol's encoding recovers the symbol. This is the arithmetic content the read phase delivers — every tape's buf field, after walking its b -cell block, holds the source cell's value.

lemma *foldl_ar_acc_encode_symbol*:
assumes *finite* Γ **and** $x \in \Gamma$
shows *foldl* (*ar_acc* Γ *bl*) (*gamma_unenum* Γ *bl* 0) (*encode_symbol* Γ *bl* x) =
 x
 $\langle proof \rangle$

The decoder's neutral element is the blank: *gamma_unenum* Γ *bl* 0 = *bl*, since the anchor puts *bl* at index 0 and the enumeration is injective.

lemma *gamma_unenum_zero*:
assumes *finite* Γ **and** $bl \in \Gamma$
shows *gamma_unenum* Γ *bl* 0 = *bl*
 $\langle proof \rangle$

An all-*BLANK*₄ block decodes to the blank: *bl* is a fixpoint of the accumulator on a *BLANK*₄ cell ($2 \cdot \text{gamma_enum } bl + 0 = 0$, decoding back to *bl*), so folding any number of *BLANK*₄ cells from *bl* stays *bl*. This is the read-phase decode correctness for the blank cell, the *BLANK*₄ counterpart of *foldl_ar_acc_encode_symbol*.

lemma *foldl_ar_acc_blank*:
assumes *finite* Γ **and** $bl \in \Gamma$
shows *foldl* (*ar_acc* Γ *bl*) *bl* (*replicate* k *BLANK*₄) = *bl*
 $\langle proof \rangle$

end
theory *AlphabetReduction_Stage*
imports *AlphabetReduction_Codec*
begin

2.6 Stage type and validity

Substep counter / phase indicator for the output machine's state set, mirroring AE's *substep_idx*. The five simulation sub-tags *AR_SimRead* / *AR_SimCompute* / *AR_SimWrite* / *AR_SimAdvance* / *AR_SimNext* mark the five phases of one M-step simulation; *AR_HaltAccept* and *AR_HaltReject* are the AR-specific terminal states that map to the substrate's $t_tm\ M''$ and $r_tm\ M''$. No validation sub-tags: AR's language theorem is quantified over encoder-image inputs only, and AR's per-tape *buf* field is populated on-the-fly during read substeps (no buffer-initialisation prelude to motivate a validation phase). Throughout, *b* abbreviates *block_width* Γ (the per-symbol cell width).

datatype *ar_substep_idx* =
AR_SimRead | *AR_SimCompute* | *AR_SimWrite* | *AR_SimAdvance*
| *AR_SimNext*
| *AR_HaltAccept* | *AR_HaltReject*

instance *ar_substep_idx* :: *finite*
 $\langle proof \rangle$

Per-tape position-kind tag, the AR analogue of AE's per-tape regime dispatch (*le0* / *le1* / *steady*), generalised to three values because of the LE-1-cell / proper-b-cell layout: *AR_AtLE* for the head at M-position 0 (the substrate-mandated single *LE4* cell), *AR_AtFirstProper* for the head at M-position 1 (immediately past the LE cell, where an *L*-move lands on the LE cell), and *AR_AtFurtherProper* for the head at M-position ≥ 2 (where an *L*-move stays in proper territory). Read by *AR_SimRead* / *AR_SimWrite* to dispatch between LE-mode and proper-mode handling, by *AR_SimAdvance* to compute the per-direction displacement, and updated by *AR_SimAdvance* per the direction.

```
datatype ar_pos_kind =
  AR_AtLE
  | AR_AtFirstProper
  | AR_AtFurtherProper
```

```
instance ar_pos_kind :: finite
<proof>
```

Stage bookkeeping carried in the output machine's state set: substep tag, current-tape index (*nat*), per-bit counter (*nat*), per-tape symbol buf (*nat* $\Rightarrow$ '*a*': holds the decoded value after read, the symbol-to-encode during write), direction-vector (*nat* $\Rightarrow$ *dir*: emitted by *AR_SimCompute*, consumed by *AR_SimAdvance*), and per-tape position-kind (*nat* $\Rightarrow$ *ar_pos_kind*: read by *SimRead*/*SimWrite* for LE/proper dispatch, updated by *SimAdvance* per the direction). The *nat* counter is loose-typed; *ar_valid_stage* below imposes the bound $i < 2 * b$ (accommodating the $2b$ -cell L-walks of *AR_SimAdvance*) to recover finiteness of the state set.

```
type_synonym 'a ar_stage =
  ar_substep_idx
   $\times$  nat
   $\times$  nat
   $\times$  (nat  $\Rightarrow$  'a)
   $\times$  (nat  $\Rightarrow$  dir)
   $\times$  (nat  $\Rightarrow$  ar_pos_kind)
```

State-level stage validity: the *nat* $\Rightarrow$ '*a* buf field stores per-tape symbols in $\Gamma \cup \{bl\}$ (so *finite* Γ bounds it), and the *nat* bit-counter is bounded by $2 * b$ (the factor 2 accommodates *AR_SimAdvance*'s L-walks of up to $2b$ cells; AR's δ' keeps the counter's substantively-reachable range within this bound, so the bound is non-restrictive on runtime states but load-bearing for finiteness of the state set). The position-kind field is unconstrained at this level — its type is already finite. Spec-side counterpart used as the stage-component restriction in *alphabet_reduce*'s output state set *Q'*, making *Q'* finite from the set-level premise *finite* Γ plus the value-level tape-count bound (*ar_stage_bounded*, next block) — without requiring *UNIV*(*nat* $\Rightarrow$ '*a*) to be a finite type. Direct AR analogue of *ae_valid_stage*.

definition *ar_valid_stage* ::
'a set $\Rightarrow$ *'a* $\Rightarrow$ *'a ar_stage* $\Rightarrow$ *bool* **where**
ar_valid_stage Γ *bl stg* $\longleftrightarrow$
 (case *stg* of ($_$, $_$, *i*, *buf*, $_$, $_$) $\Rightarrow$
 $i < 2 * \text{block_width } \Gamma \wedge (\forall k. \text{buf } k \in \Gamma \cup \{bl\}))$)

Tape count bound: the value-level isolation of the tape-count fact. The per-tape function fields freeze to their initial values beyond the active tape count K (*buf* $\rightarrow$ *bl*, *dvec* $\rightarrow$ *dir.N*, *posk* $\rightarrow$ *AR_AtLE*), and the current-tape scalar *tk* stays $\leq K$. Kept *separate* from *ar_valid_stage* (the per-step content invariant threaded through the simulation): the bound is needed only by *alphabet_reduce*'s *Q'*, *finite_ar_valid_stages*, the three phase combinators, and a single substep-union preservation step — not at the 120 content sites — so the tape count fact is localised rather than smeared. (AE's *ae_valid_stage* folds the bound in directly; that ripple is slated for de-rippling in AE's pre-AFP defensive review, converging on this isolated shape.)

definition *ar_stage_bounded* ::
'a $\Rightarrow$ *nat* $\Rightarrow$ *'a ar_stage* $\Rightarrow$ *bool* **where**
ar_stage_bounded *bl K stg* $\longleftrightarrow$
 (case *stg* of ($_$, *tk*, $_$, *buf*, *dvec*, *posk*) $\Rightarrow$
 $tk < K$
 $\wedge (\forall j \geq K. \text{buf } j = bl)$
 $\wedge (\forall j \geq K. \text{dvec } j = \text{dir.N})$
 $\wedge (\forall j \geq K. \text{posk } j = \text{AR_AtLE}))$)

lemma *finite_ar_valid_stages*:
fixes $\Gamma :: 'a \text{ set}$ **and** $bl :: 'a$ **and** $K :: nat$
assumes *finG*: *finite* Γ
shows *finite* {*stg* :: *'a ar_stage*.
 $\text{ar_valid_stage } \Gamma \text{ bl stg} \wedge \text{ar_stage_bounded bl K stg}$ }
 <proof>

2.7 Per-tape enumeration helpers

Per-tape walk helpers. Under value-level tape count a tape index is a natural number directly (tape *kk* sits at position *kk*, the active tapes being $\{.. < k_tm \text{ } M\}$), so the abstract enumeration of the old *'k* tape-index type collapses: *k_idx* and *k_unidx* are the identity and *k_succ* is *Suc*. They are retained as thin wrappers through the tape count migration (inlined in the cleanup sweep) so the per-phase walk lemmas keep their shape. The per-tape substep relations (*ar_delta_read*, *ar_delta_write*, *ar_delta_advance*) walk tapes in order, processing tape *kk*, advancing to *k_succ kk*, and leaving the phase when *is_last_k M kk* (*Suc kk* = *k_tm M*) fires.

definition *k_idx* :: *nat* $\Rightarrow$ *nat* **where**
 $k_idx \text{ } kk = kk$

definition *k_unidx* :: *nat* $\Rightarrow$ *nat* **where**

$$k_unidx\ j = j$$

definition $k_succ :: nat \Rightarrow nat$ **where**
 $k_succ\ kk = Suc\ kk$

definition $is_last_k :: ('q, 'a)\ mttm \Rightarrow nat \Rightarrow bool$ **where**
 $is_last_k\ M\ kk \longleftrightarrow Suc\ kk = k_tm\ M$

Navigation facts the per-phase tape walks consume. Under the identity collapse these are immediate: k_idx and k_unidx fix 0 and k_idx is injective, the successor of $k_unidx\ j$ is $k_unidx\ (Suc\ j)$, and $is_last_k\ M\ (k_unidx\ j)$ reduces to $Suc\ j = k_tm\ M$. The bounded hypotheses are retained so existing call sites keep their shape across the migration.

lemma $k_idx_zero: k_idx\ 0 = 0$
 $\langle proof \rangle$

lemma $k_unidx_zero: k_unidx\ 0 = 0$
 $\langle proof \rangle$

lemma $k_idx_inj: inj_on\ k_idx\ UNIV$
 $\langle proof \rangle$

lemma $k_idx_unidx:$
assumes $j < k_tm\ M$
shows $k_idx\ (k_unidx\ j) = j$
 $\langle proof \rangle$

lemma $k_succ_unidx:$
assumes $Suc\ j < k_tm\ M$
shows $k_succ\ (k_unidx\ j) = k_unidx\ (Suc\ j)$
 $\langle proof \rangle$

lemma $is_last_k_unidx:$
assumes $j < k_tm\ M$
shows $is_last_k\ M\ (k_unidx\ j) \longleftrightarrow Suc\ j = k_tm\ M$
 $\langle proof \rangle$

2.8 Per-substep dispatch helpers

Displacement of an $AR_SimAdvance$ walk on one tape, dispatched on the tape's current direction (from $dvec$) and position-kind. Encodes the displacement table: zero for R (head already at $sim_pos\ (p+1)$ after the per-tape write phase), one for N from AR_AtLE (single L -move back to position 0), b for N from proper (back to $sim_pos\ p$), $Suc\ b$ for L from $AR_AtFirstProper$ (back to position 0), and $2 \cdot b$ for L from $AR_AtFurtherProper$ (back to $sim_pos\ (p-1)$). The L -from- AR_AtLE case is forbidden by the substrate's δLE invariant; setting its displacement to 0 here makes the advance relation exclude that case naturally (no stepping arm is satisfied, no boundary

condition fires).

```
fun ar_disp :: nat  $\Rightarrow$  dir  $\Rightarrow$  ar_pos_kind  $\Rightarrow$  nat where
  ar_disp _ dir.R _ = 0
| ar_disp _ dir.N AR_AtLE = 1
| ar_disp k dir.N AR_AtFirstProper = k
| ar_disp k dir.N AR_AtFurtherProper = k
| ar_disp _ dir.L AR_AtLE = 0
| ar_disp k dir.L AR_AtFirstProper = Suc k
| ar_disp k dir.L AR_AtFurtherProper = 2 * k
```

Per-tape position-kind transition under *AR_SimAdvance*, dispatched on (direction, pre-advance position-kind). Encodes the transition table: *R*-advance bumps position-kind up the ladder (*AR_AtLE* $\rightarrow$ *AR_AtFirstProper* $\rightarrow$ *AR_AtFurtherProper*; the last is a fixed point); *N*-advance preserves position-kind; *L*-advance from *AR_AtFirstProper* drops to *AR_AtLE*; *L*-advance from *AR_AtFurtherProper* remains *AR_AtFurtherProper* as a defensive default — the actual post-advance position can be either *AR_AtFirstProper* (if $p - 1 = 1$) or *AR_AtFurtherProper* (otherwise), and a one-cell look-back substep at the next *AR_SimRead* refines this distinction. The *L*-from-*AR_AtLE* case is forbidden by δLE and is never substantively reached; the dummy mapping to *AR_AtLE* below keeps *ar_newpos* total.

```
fun ar_newpos :: dir  $\Rightarrow$  ar_pos_kind  $\Rightarrow$  ar_pos_kind where
  ar_newpos dir.R AR_AtLE = AR_AtFirstProper
| ar_newpos dir.R AR_AtFirstProper = AR_AtFurtherProper
| ar_newpos dir.R AR_AtFurtherProper = AR_AtFurtherProper
| ar_newpos dir.N AR_AtLE = AR_AtLE
| ar_newpos dir.N AR_AtFirstProper = AR_AtFirstProper
| ar_newpos dir.N AR_AtFurtherProper = AR_AtFurtherProper
| ar_newpos dir.L AR_AtLE = AR_AtLE
| ar_newpos dir.L AR_AtFirstProper = AR_AtLE
| ar_newpos dir.L AR_AtFurtherProper = AR_AtFurtherProper
```

The cell-level write image for a per-tape symbol in the proper region. Returns the j -th sym4 cell of the symbol's image: *BLANK4* for every position if the symbol is the blank *bl* (the blank's cell-repr is b consecutive blanks); the j -th bit of *encode_symbol* Γ *bl* x otherwise. Used by *ar_delta_write*'s proper-arm forward-write phase to emit one cell per substep. The *le_tm M* case is excluded — *ar_delta_write*'s LE-arm short-circuits writes for that symbol, so the head never enters the forward-write phase with *buf tk* = *le_tm M*.

```
definition write_bit ::
  'a set  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  nat  $\Rightarrow$  sym4 where
  write_bit  $\Gamma$  bl x j =
    (if x = bl then BLANK4 else encode_symbol  $\Gamma$  bl x ! j)
```

end

```

theory AlphabetReduction_Delta
imports AlphabetReduction_Stage
begin

```

2.9 Substep transition relations

The output machine's δ' is defined as a union of five per-substep transition relations, mirroring AE's *alphabet_enlarge_delta* shape. Each substep relation is a set of substrate-shape 5-tuples $((q, stg), a, (q', stg'), a', d)$ where q ranges over $Q_tm\ M$, stg over $'a\ ar_stage$, a / a' over $nat \Rightarrow sym4$, and d over $nat \Rightarrow dir$. The five substep relations are defined below, one per simulation phase.

No validation cluster: AR's language theorem is quantified over encoder-image inputs only, so non-canonical *sym4* inputs are outside the theorem's scope and M's behaviour on them is unconstrained.

The union is intersected with two global restrictions: LE-preservation (no transition forges *LE4* out of a non-*LE4* cell, matching the substrate's δLE invariant) and *ar_valid_stage*-membership on both source and target stages (matching the non-product Q' shape). Throughout, b abbreviates *block_width* Γ (the per-symbol cell width).

definition *ar_delta_read* ::

```

('q, 'a) mttm
 $\Rightarrow$  (('q  $\times$  'a ar_stage)
 $\times$  (nat  $\Rightarrow$  sym4)
 $\times$  ('q  $\times$  'a ar_stage)
 $\times$  (nat  $\Rightarrow$  sym4)
 $\times$  (nat  $\Rightarrow$  dir)) set where

```

ar_delta_read $M =$

— LE-arm, non-last tape: head at position 0 reads *LE4*, sets *buf tk* := *le_tm M* directly (no accumulator needed for the single LE-cell), advances R on *tk* to position 1, and transitions to the next tape's read. *posk tk* stays *AR_AtLE* throughout this substep — *SimAdvance* will update it later when the head leaves the LE region.

```

{((q, AR_SimRead, tk, 0, buf, dvec, posk), a,
  (q, AR_SimRead, k_succ tk, 0,
    buf(tk := le_tm M), dvec, posk), a, d) |
  q tk buf dvec posk a d.
  q  $\in$  Q_tm M
   $\wedge$   $\neg$  is_last_k M tk
   $\wedge$  posk tk = AR_AtLE
   $\wedge$  a tk = LE4
   $\wedge$  d = ( $\lambda kk.$  if  $kk = tk$  then dir.R else dir.N)}
```

$\cup$

— LE-arm, last tape: same single-substep LE-read, transitions to *AR_SimCompute* instead of the next tape.

```

{((q, AR_SimRead, tk, 0, buf, dvec, posk), a,
  (q, AR_SimCompute, k_unidx 0, 0,
    buf(tk := le_tm M), dvec, posk), a, d) |
```

$q \text{ tk buf dvec posk } a \text{ d.}$
 $q \in Q_tm \ M$
 $\wedge is_last_k \ M \ tk$
 $\wedge posk \ tk = AR_AtLE$
 $\wedge a \ tk = LE4$
 $\wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.R \text{ else } dir.N)\}$

$\cup$
 — Proper-arm look-back step 1 ($i = 0$): head at $sim_pos(p)$ moves L on tk to $sim_pos(p) - 1$. The cell at $sim_pos(p)$ is some $BIT0$ / $BIT1$ / $BLANK4$ (not $LE4$; enforced as a δLE precondition). Buf, posk, dvec unchanged; substep transitions to $i = 1$.

$\{((q, AR_SimRead, tk, 0, buf, dvec, posk), a,$
 $(q, AR_SimRead, tk, Suc \ 0, buf, dvec, posk), a, d) \mid$
 $q \text{ tk buf dvec posk } a \text{ d.}$
 $q \in Q_tm \ M$
 $\wedge posk \ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
 $\wedge a \ tk \neq LE4$
 $\wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.L \text{ else } dir.N)\}$

$\cup$
 — Proper-arm look-back step 2 ($i = 1$): head at $sim_pos(p) - 1$ reads the cell there; if $LE4$, the head was at $sim_pos \ 1 = 1$ in the previous step so refine $posk \ tk := AR_AtFirstProper$; otherwise refine to $AR_AtFurtherProper$. Reset $buf \ tk := gamma_unenum \ \Gamma \ bl \ 0$ (the partial-decoded “0 bits read so far” symbol) to prepare for the per-bit accumulator below. R -move back to $sim_pos(p)$; δLE with $a \ tk = LE4$ is fine because the direction is R . Transitions to $i = 2$.

$\{((q, AR_SimRead, tk, Suc \ 0, buf, dvec, posk), a,$
 $(q, AR_SimRead, tk, Suc \ (Suc \ 0),$
 $buf(tk := gamma_unenum \ (\Gamma_tm \ M) \ (bl_tm \ M) \ 0),$
 $dvec,$
 $posk(tk := \text{ if } a \ tk = LE4 \text{ then } AR_AtFirstProper$
 $\text{ else } AR_AtFurtherProper)), a, d) \mid$
 $q \text{ tk buf dvec posk } a \text{ d.}$
 $q \in Q_tm \ M$
 $\wedge posk \ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
 $\wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.R \text{ else } dir.N)\}$

$\cup$
 — Proper-arm per-bit stepping ($2 \leq i \leq b$): head at $sim_pos(p) + (i - 2)$, read cell, accumulate bit_value into the partial-decoded $buf \ tk$ via the $gamma_enum / gamma_unenum$ roundtrip $partial' = 2 \cdot gamma_enum \ (buf \ tk) + bit_value \ (a \ tk)$, $buf' \ tk := gamma_unenum \ partial'$. R -move on tk . Transitions to $i + 1$ within $AR_SimRead$; the next per-bit substep continues the accumulator chain.

$\{((q, AR_SimRead, tk, i, buf, dvec, posk), a,$
 $(q, AR_SimRead, tk, Suc \ i,$
 $buf(tk := gamma_unenum \ (\Gamma_tm \ M) \ (bl_tm \ M)$
 $(2 * gamma_enum \ (\Gamma_tm \ M) \ (bl_tm \ M) \ (buf \ tk)$
 $+ bit_value \ (a \ tk))),$
 $dvec, posk), a, d) \mid$
 $q \text{ tk } i \text{ buf dvec posk } a \text{ d.}$
 $q \in Q_tm \ M$

$$\begin{aligned}
& \wedge \text{posk } tk \in \{AR_AtFirstProper, AR_AtFurtherProper\} \\
& \wedge 2 \leq i \\
& \wedge \text{Suc } i \leq \text{Suc } (\text{block_width } (\Gamma_tm \ M)) \\
& \wedge d = (\lambda kk. \text{if } kk = tk \text{ then } dir.R \text{ else } dir.N) \} \\
& \cup \\
& \text{— Proper-arm per-bit boundary } (i = \text{Suc } (b)), \text{ non-last tape: same accumulator} \\
& \text{step as above on the last bit } (j = b - 1), \text{ buf' } tk \text{ holds the fully-decoded } M\text{-symbol} \\
& (\in \Gamma \text{ for valid encoder images; falls back to } bl_tm \ M \text{ for non-encoder inputs), then} \\
& \text{transitions to the same phase on } k_succ \ tk \text{ with bit-counter reset to } 0. \\
& \{((q, AR_SimRead, tk, i, buf, dvec, posk), a, \\
& \quad (q, AR_SimRead, k_succ \ tk, 0, \\
& \quad \quad \text{buf}(tk := \text{gamma_unenum } (\Gamma_tm \ M) (bl_tm \ M) \\
& \quad \quad \quad (2 * \text{gamma_enum } (\Gamma_tm \ M) (bl_tm \ M) (buf \ tk) \\
& \quad \quad \quad + \text{bit_value } (a \ tk))), \\
& \quad dvec, posk), a, d) \mid \\
& q \ tk \ i \ buf \ dvec \ posk \ a \ d. \\
& q \in Q_tm \ M \\
& \wedge \neg is_last_k \ M \ tk \\
& \wedge \text{posk } tk \in \{AR_AtFirstProper, AR_AtFurtherProper\} \\
& \wedge i = \text{Suc } (\text{block_width } (\Gamma_tm \ M)) \\
& \wedge d = (\lambda kk. \text{if } kk = tk \text{ then } dir.R \text{ else } dir.N) \} \\
& \cup \\
& \text{— Proper-arm per-bit boundary, last tape: same last-bit accumulator step,} \\
& \text{transitions to } AR_SimCompute \text{ with current-tape reset to } k_unidx \ 0. \\
& \{((q, AR_SimRead, tk, i, buf, dvec, posk), a, \\
& \quad (q, AR_SimCompute, k_unidx \ 0, 0, \\
& \quad \quad \text{buf}(tk := \text{gamma_unenum } (\Gamma_tm \ M) (bl_tm \ M) \\
& \quad \quad \quad (2 * \text{gamma_enum } (\Gamma_tm \ M) (bl_tm \ M) (buf \ tk) \\
& \quad \quad \quad + \text{bit_value } (a \ tk))), \\
& \quad dvec, posk), a, d) \mid \\
& q \ tk \ i \ buf \ dvec \ posk \ a \ d. \\
& q \in Q_tm \ M \\
& \wedge is_last_k \ M \ tk \\
& \wedge \text{posk } tk \in \{AR_AtFirstProper, AR_AtFurtherProper\} \\
& \wedge i = \text{Suc } (\text{block_width } (\Gamma_tm \ M)) \\
& \wedge d = (\lambda kk. \text{if } kk = tk \text{ then } dir.R \text{ else } dir.N) \}
\end{aligned}$$

definition *ar_delta_compute* ::

$$\begin{aligned}
& ('q, 'a) \text{ mttm} \\
& \Rightarrow (('q \times 'a \ ar_stage) \\
& \quad \times (\text{nat} \Rightarrow \text{sym4}) \\
& \quad \times ('q \times 'a \ ar_stage) \\
& \quad \times (\text{nat} \Rightarrow \text{sym4}) \\
& \quad \times (\text{nat} \Rightarrow \text{dir})) \text{ set where}
\end{aligned}$$

$$\begin{aligned}
& ar_delta_compute \ M = \\
& \{((q, AR_SimCompute, tk, i, buf, dvec, posk), a, \\
& \quad (q', AR_SimWrite, 0, 0, m_a', m_d, posk), a, d) \mid \\
& q \ buf \ q' \ m_a' \ m_d \ tk \ i \ dvec \ posk \ a \ d. \\
& (q, buf, q', m_a', m_d) \in \text{delta_tm } M
\end{aligned}$$

$$\wedge d = (\lambda_{-}. \text{dir}.N)\}$$

— One δ' -tuple per M 's δ -tuple. Matches the source-stage *buf* field (the decoded per-tape symbol vector, produced by *AR_SimRead*) against the M-side read symbol vector of the M - δ -tuple ($q, \text{buf}, q', m_a', m_d$), and threads the M-side output: new M-state q' , write-back vector m_a' (next phase's per-tape symbols to encode), and direction-vector m_d (consumed by *AR_SimAdvance*). The substrate-side read a and write $a' = a$ are unconstrained at this substep (no head moves, no writes); direction is constant N . The per-tape *posk* carries through unchanged (head positions don't change during compute). No halt-state shortcut: M-side halt detection is centralised at *ar_delta_next*'s three-arm dispatch.

definition *ar_delta_write* ::

$$\begin{aligned} &('q, 'a) \text{ mttm} \\ &\Rightarrow (('q \times 'a \text{ ar_stage}) \\ &\quad \times (\text{nat} \Rightarrow \text{sym4}) \\ &\quad \times ('q \times 'a \text{ ar_stage}) \\ &\quad \times (\text{nat} \Rightarrow \text{sym4}) \\ &\quad \times (\text{nat} \Rightarrow \text{dir})) \text{ set where} \\ &\text{ar_delta_write } M = \end{aligned}$$

— LE-arm, non-last tape: *posk tk = AR_AtLE* means tape tk 's head sits on the reduced boundary (*sim_pos 0 = 0*), whose *LE4* cell need not be rewritten (δLE requires writing *le* back, which is exactly the present cell). Keyed on the position-kind flag, not on *buf tk = le_tm M*, so a source machine that writes *le* off the boundary (a tape cut) is handled by the proper arm below, as ordinary data. Skip the per-tape write phase entirely: single substep with all directions N , transitioning to the same phase on the next tape with bit-counter still at 0 .

$$\begin{aligned} &\{((q, \text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk}), a, \\ &\quad (q, \text{AR_SimWrite}, k_succ\ tk, 0, \text{buf}, \text{dvec}, \text{posk}), a, d) \mid \\ &\quad q\ tk\ \text{buf}\ \text{dvec}\ \text{posk}\ a\ d. \\ &\quad q \in Q_tm\ M \\ &\quad \wedge \text{posk}\ tk = \text{AR_AtLE} \\ &\quad \wedge \neg \text{is_last_k}\ M\ tk \\ &\quad \wedge d = (\lambda_{-}. \text{dir}.N)\} \end{aligned}$$

$\cup$

— LE-arm, last tape: transitions to *AR_SimAdvance* with the current-tape index reset to *k_unidx 0*.

$$\begin{aligned} &\{((q, \text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk}), a, \\ &\quad (q, \text{AR_SimAdvance}, k_unidx\ 0, 0, \text{buf}, \text{dvec}, \text{posk}), a, d) \mid \\ &\quad q\ tk\ \text{buf}\ \text{dvec}\ \text{posk}\ a\ d. \\ &\quad q \in Q_tm\ M \\ &\quad \wedge \text{posk}\ tk = \text{AR_AtLE} \\ &\quad \wedge \text{is_last_k}\ M\ tk \\ &\quad \wedge d = (\lambda_{-}. \text{dir}.N)\} \end{aligned}$$

$\cup$

— Proper-arm, back-walk phase ($0 \leq i < b$): head moves L on tk , N elsewhere; no writes. At $i = b - 1$ the substep transitions to $i = b$, starting the forward-write phase below. Both phases share the *AR_SimWrite* substep tag; the bit-counter distinguishes them. $a\ tk \neq LE4$ enforced for δLE .

$$\{((q, \text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk}), a,$$

$(q, AR_SimWrite, tk, Suc\ i, buf, dvec, posk), a, d) \mid$
 $q\ tk\ i\ buf\ dvec\ posk\ a\ d.$
 $q \in Q_tm\ M$
 $\wedge posk\ tk \neq AR_AtLE$
 $\wedge a\ tk \neq LE4$
 $\wedge Suc\ i \leq block_width\ (\Gamma_tm\ M)$
 $\wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.L \text{ else } dir.N)\}$
 $\cup$
 — Proper-arm, forward-write stepping phase ($b \leq i < 2b - 1$): write $write_bit$
 $\Gamma\ bl\ (buf\ tk)\ (i - b)$ at tk , R on tk , N elsewhere. All other tapes have their cells
 preserved. The bit-counter advances; the next substep continues forward-write.
 $\{((q, AR_SimWrite, tk, i, buf, dvec, posk), a,$
 $(q, AR_SimWrite, tk, Suc\ i, buf, dvec, posk), a', d) \mid$
 $q\ tk\ i\ buf\ dvec\ posk\ a\ a'\ d.$
 $q \in Q_tm\ M$
 $\wedge posk\ tk \neq AR_AtLE$
 $\wedge block_width\ (\Gamma_tm\ M) \leq i$
 $\wedge Suc\ i < 2 * block_width\ (\Gamma_tm\ M)$
 $\wedge a' = (\lambda kk. \text{ if } kk = tk$
 $\quad \text{ then } write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad \quad (buf\ tk)$
 $\quad \quad (i - block_width\ (\Gamma_tm\ M))$
 $\quad \text{ else } a\ kk)$
 $\wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.R \text{ else } dir.N)\}$
 $\cup$
 — Proper-arm, forward-write boundary ($Suc\ i = 2b$), non-last tape: last cell of
 $cell_repr\ (buf\ tk)$ is written, head moves R on tk , transitions to the same phase on
 $k_succ\ tk$ with bit-counter reset. Cells on other tapes preserved.
 $\{((q, AR_SimWrite, tk, i, buf, dvec, posk), a,$
 $(q, AR_SimWrite, k_succ\ tk, 0, buf, dvec, posk), a', d) \mid$
 $q\ tk\ i\ buf\ dvec\ posk\ a\ a'\ d.$
 $q \in Q_tm\ M$
 $\wedge posk\ tk \neq AR_AtLE$
 $\wedge \neg is_last_k\ M\ tk$
 $\wedge Suc\ i = 2 * block_width\ (\Gamma_tm\ M)$
 $\wedge a' = (\lambda kk. \text{ if } kk = tk$
 $\quad \text{ then } write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad \quad (buf\ tk)$
 $\quad \quad (i - block_width\ (\Gamma_tm\ M))$
 $\quad \text{ else } a\ kk)$
 $\wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.R \text{ else } dir.N)\}$
 $\cup$
 — Proper-arm, forward-write boundary, last tape: same last-cell write as above,
 transitions to $AR_SimAdvance$ with the current-tape index reset to $k_unidx\ 0$.
 $\{((q, AR_SimWrite, tk, i, buf, dvec, posk), a,$
 $(q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk), a', d) \mid$
 $q\ tk\ i\ buf\ dvec\ posk\ a\ a'\ d.$
 $q \in Q_tm\ M$
 $\wedge posk\ tk \neq AR_AtLE$

$$\begin{aligned}
& \wedge is_last_k\ M\ tk \\
& \wedge Suc\ i = 2 * block_width\ (\Gamma_tm\ M) \\
& \wedge a' = (\lambda kk. \text{ if } kk = tk \\
& \quad \text{ then write_bit } (\Gamma_tm\ M)\ (bl_tm\ M) \\
& \quad \quad (buf\ tk) \\
& \quad \quad (i - block_width\ (\Gamma_tm\ M)) \\
& \quad \text{ else } a\ kk) \\
& \wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.R \text{ else } dir.N) \}
\end{aligned}$$

definition *ar_delta_advance* ::

$$\begin{aligned}
& ('q, 'a)\ mttm \\
& \Rightarrow (('q \times 'a\ ar_stage) \\
& \quad \times (nat \Rightarrow sym4) \\
& \quad \times ('q \times 'a\ ar_stage) \\
& \quad \times (nat \Rightarrow sym4) \\
& \quad \times (nat \Rightarrow dir))\ set\ \mathbf{where}
\end{aligned}$$

$$ar_delta_advance\ M =$$

— Stepping arm: head moves L on tape tk , N elsewhere; stays in $AR_SimAdvance$ with the bit-counter advanced. Active when $Suc\ i$ is strictly below the per-tape displacement $ar_disp\ b\ (dvec\ tk)\ (posk\ tk)$; the R - and L -from- AR_AtLE cases have displacement 0 , so no stepping tuple fires; for N -from- AR_AtLE the displacement is 1 , so stepping never fires and the single substep goes through one of the boundary arms. The read symbol on tk must not be $LE4$ (δLE : an L -move from a tape reading $LE4$ is forbidden).

$$\begin{aligned}
& \{((q, AR_SimAdvance, tk, i, buf, dvec, posk), a, \\
& \quad (q, AR_SimAdvance, tk, Suc\ i, buf, dvec, posk), a, d) \mid \\
& \quad q\ tk\ i\ buf\ dvec\ posk\ a\ d. \\
& \quad q \in Q_tm\ M \\
& \quad \wedge a\ tk \neq LE4 \\
& \quad \wedge Suc\ i < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk) \\
& \quad \wedge d = (\lambda kk. \text{ if } kk = tk \text{ then } dir.L \text{ else } dir.N) \}
\end{aligned}$$

$\cup$

— Boundary arm, non-last tape. Two firing sub-cases: the R -direction case (zero displacement, no L -move; $i = 0$ and the entire direction vector is N), and the non- R case at $Suc\ i$ equal to the displacement (one last L -move on tk ; $a\ tk \neq LE4$ enforced for δLE). Both sub-cases transition to the same phase on the next tape $k_succ\ tk$ (bit-counter reset to 0) with $posk\ tk$ updated by ar_newpos ; all other tapes' $posk$ entries are preserved.

$$\begin{aligned}
& \{((q, AR_SimAdvance, tk, i, buf, dvec, posk), a, \\
& \quad (q, AR_SimAdvance, k_succ\ tk, 0, buf, dvec, \\
& \quad \quad posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk))), a, d) \mid \\
& \quad q\ tk\ i\ buf\ dvec\ posk\ a\ d. \\
& \quad q \in Q_tm\ M \\
& \quad \wedge \neg is_last_k\ M\ tk \\
& \quad \wedge ((dvec\ tk = dir.R \wedge i = 0) \\
& \quad \quad \vee (dvec\ tk \neq dir.R \\
& \quad \quad \wedge a\ tk \neq LE4 \\
& \quad \quad \wedge Suc\ i = ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)))
\end{aligned}$$

$$\wedge d = (\lambda kk. \text{ if } kk = tk \wedge dvec\ tk \neq dir.R \\ \text{ then } dir.L \text{ else } dir.N) \}$$

∪

— Boundary arm, last tape. Same firing sub-cases as the non-last-tape arm; transitions to $AR_SimNext$ instead of advancing tk . The current-tape field is reset to $k_unidx\ 0$ (the first tape in the enumeration; matches $AR_SimNext$'s starting convention and the next $AR_SimRead$'s initial tape).

$$\{((q, AR_SimAdvance, tk, i, buf, dvec, posk), a, \\ (q, AR_SimNext, k_unidx\ 0, 0, buf, dvec, \\ posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk))), a, d) \mid \\ q\ tk\ i\ buf\ dvec\ posk\ a\ d. \\ q \in Q_tm\ M \\ \wedge is_last_k\ M\ tk \\ \wedge ((dvec\ tk = dir.R \wedge i = 0) \\ \vee (dvec\ tk \neq dir.R \\ \wedge a\ tk \neq LE) \\ \wedge Suc\ i = ar_disp\ (block_width\ (\Gamma_tm\ M)) \\ (dvec\ tk)\ (posk\ tk))) \\ \wedge d = (\lambda kk. \text{ if } kk = tk \wedge dvec\ tk \neq dir.R \\ \text{ then } dir.L \text{ else } dir.N) \}$$

definition $ar_delta_next ::$

$$('q, 'a)\ mttm \\ \Rightarrow (('q \times 'a\ ar_stage) \\ \times (nat \Rightarrow sym4) \\ \times ('q \times 'a\ ar_stage) \\ \times (nat \Rightarrow sym4) \\ \times (nat \Rightarrow dir))\ \text{set where}$$

$$ar_delta_next\ M =$$

$$\{((q, AR_SimNext, tk, i, buf, dvec, posk), a, \\ (q, AR_SimRead, 0, 0, buf, dvec, posk), a, d) \mid \\ q\ tk\ i\ buf\ dvec\ posk\ a\ d. \\ q \in Q_tm\ M \wedge q \neq t_tm\ M \wedge q \neq r_tm\ M \\ \wedge d = (\lambda_ . dir.N) \}$$

∪

$$\{((q, AR_SimNext, tk, i, buf, dvec, posk), a, \\ (q, AR_HaltAccept, 0, 0, \\ (\lambda_ . bl_tm\ M), (\lambda_ . dir.N), (\lambda_ . AR_AtLE))), a, d) \mid \\ q\ tk\ i\ buf\ dvec\ posk\ a\ d. \\ q = t_tm\ M \\ \wedge d = (\lambda_ . dir.N) \}$$

∪

$$\{((q, AR_SimNext, tk, i, buf, dvec, posk), a, \\ (q, AR_HaltReject, 0, 0, \\ (\lambda_ . bl_tm\ M), (\lambda_ . dir.N), (\lambda_ . AR_AtLE))), a, d) \mid \\ q\ tk\ i\ buf\ dvec\ posk\ a\ d. \\ q = r_tm\ M \\ \wedge d = (\lambda_ . dir.N) \}$$

— End-of-M-step handshake. Three arms dispatch on the M-state q : non-terminal

routes to $AR_SimRead$ for the next M-step (with bit-counter and current-tape both reset to 0); $q = t_tm\ M$ routes to the canonical accept stage; $q = r_tm\ M$ routes to the canonical reject stage. The non-terminal arm preserves buf , the direction-vector field $dvec$ (residual from the just-completed M-step; no longer consulted), and the per-tape position-kind $posk$ (carries the head-region across M-steps so the next $AR_SimRead$'s LE-vs-proper dispatch sees the correct value). The two terminal arms instead reset $buf / dvec / posk$ to the canonical halt values ($\lambda_ . bl_tm\ M / \lambda_ . dir.N / \lambda_ . AR_AtLE$), so the target stage is exactly $ar_accept_stage (bl_tm\ M) / ar_reject_stage (bl_tm\ M)$ and the handshake lands on M' 's halt state $t_tm\ M' / r_tm\ M'$. $Lang_mttm$ matches the full mt_state (tape and heads free), so this canonical landing is what makes M' acceptance / rejection detectable; the reset is semantically free (the residual fields are never read after halting). The direction vector is all N (no head moves at the handshake); cell vectors $a' = a$ (no writes), so LE-preservation passes trivially.

The output machine's full transition relation: union of the five per-substep relations, intersected with three global restrictions — δLE -backward (no transition forges $LE4$ out of a non- $LE4$ cell), δLE -forward (every transition reading $LE4$ on tape k rewrites $LE4$ back on the same tape and moves N or R), and ar_valid_stage -membership on both source and target stages. The two δLE filters together discharge the substrate's bidirectional δLE invariant uniformly — without per-arm $a\ tk \neq LE4$ preconditions on every arm that performs a write distinct from the read cell (proper-arm forward-write in particular). The per-arm $a\ tk \neq LE4$ constraints that do appear (in look-back step 1, back-walk, and advance L-step) reflect the simulation invariant rather than the bare δLE requirement: those arms move L on the current tape, which the substrate forbids from an $LE4$ cell regardless of write content.

definition $alphabet_reduce_delta ::$

$$\begin{aligned}
& ('q, 'a)\ mttm \\
& \Rightarrow (('q \times 'a\ ar_stage) \\
& \quad \times (nat \Rightarrow sym4) \\
& \quad \times ('q \times 'a\ ar_stage) \\
& \quad \times (nat \Rightarrow sym4) \\
& \quad \times (nat \Rightarrow dir))\ set\ \textbf{where} \\
& alphabet_reduce_delta\ M = \\
& \quad (ar_delta_read\ M \cup ar_delta_compute\ M \\
& \quad \cup ar_delta_write\ M \cup ar_delta_advance\ M \\
& \quad \cup ar_delta_next\ M) \\
& \quad \cap \{(s, a, s', a', d). \\
& \quad \quad \forall k. a'k = LE4 \longrightarrow ak = LE4\} \\
& \quad \cap \{(s, a, s', a', d). \\
& \quad \quad \forall k. ak = LE4 \longrightarrow a'k = LE4 \wedge dk \in \{dir.N, dir.R\}\} \\
& \quad \cap \{(s, a, s', a', d). \\
& \quad \quad ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (snd\ s) \\
& \quad \quad \wedge ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (snd\ s')\} \\
& \quad \cap \{(s, a, s', a', d). \\
& \quad \quad \forall j \geq k_tm\ M. aj = BLANK4 \wedge a'j = BLANK4 \wedge dj = dir.N\}
\end{aligned}$$

$$\cap \{(s, a, s', a', d). \\ ar_stage_bounded (bl_tm M) (k_tm M) (snd s) \\ \wedge ar_stage_bounded (bl_tm M) (k_tm M) (snd s')\}$$

Initial / halt stages used to populate the output machine's $s' / t' / r'$ components. All three share the same shape: substep-counter $i = 0$, current-tape 0 placeholder (per-tape phase fields are inactive in $AR_SimRead$'s initial entry and inactive in halt states), per-tape *buf* initialised to bl (any value in $\Gamma \cup \{\!\!|bl|\!\!\}$ would satisfy ar_valid_stage ; bl is the canonical placeholder), per-tape direction-vector $dir.N$ (irrelevant outside $AR_SimAdvance$), per-tape position-kind AR_AtLE (the initial head position is 0 ; for halt states the field is vestigial). The three stages differ only in their $ar_substep_idx$ tag.

definition $ar_init_stage ::$
 $'a \Rightarrow 'a\ ar_stage$ **where**
 $ar_init_stage\ bl =$
 $(AR_SimRead, 0, 0, (\lambda_ . bl), (\lambda_ . dir.N), (\lambda_ . AR_AtLE))$

definition $ar_accept_stage ::$
 $'a \Rightarrow 'a\ ar_stage$ **where**
 $ar_accept_stage\ bl =$
 $(AR_HaltAccept, 0, 0, (\lambda_ . bl), (\lambda_ . dir.N), (\lambda_ . AR_AtLE))$

definition $ar_reject_stage ::$
 $'a \Rightarrow 'a\ ar_stage$ **where**
 $ar_reject_stage\ bl =$
 $(AR_HaltReject, 0, 0, (\lambda_ . bl), (\lambda_ . dir.N), (\lambda_ . AR_AtLE))$

Positivity of the block width: b is bounded below by 1 via the *max* on the right-hand side. Used below to discharge the $i < 2 \cdot b$ conjunct of ar_valid_stage at $i = 0$, and downstream by the per-substep step-count lemmas.

lemma $block_width_pos: 1 \leq block_width\ \Gamma$
 $\langle proof \rangle$

The three stage constants used to populate $s', t',$ and r' all satisfy $ar_valid_stage\ \Gamma\ bl$ for any Γ and bl : the bit-counter is $0 < 2 \cdot b$ (since $b \geq 1$) and the *buf* field is constantly $bl \in \Gamma \cup \{\!\!|bl|\!\!\}$.

lemma $ar_valid_stage_init: ar_valid_stage\ \Gamma\ bl\ (ar_init_stage\ bl)$
 $\langle proof \rangle$

lemma $ar_valid_stage_accept: ar_valid_stage\ \Gamma\ bl\ (ar_accept_stage\ bl)$
 $\langle proof \rangle$

lemma $ar_valid_stage_reject: ar_valid_stage\ \Gamma\ bl\ (ar_reject_stage\ bl)$
 $\langle proof \rangle$

The alphabet-reduction combinator. Input: $mttm$ over $'a$, with tape alphabet $\Gamma_tm\ M$ a finite subset of $'a$ (via $valid_mttm$) of cardinality ≥ 4 .

Output: $mttm$ over sym_4 , with state set $'q \times 'a \text{ ar_stage}$. Tape count k_tm M is preserved.

Output components:

- $Q' = Q_M \times \{\text{stg. ar_valid_stage } \Gamma_M \text{ bl_M stg}\}$ (the non-product Q-shape);
- $\Sigma' = \{\text{BIT0}, \text{BIT1}\}$ (the encoded input alphabet);
- $\Gamma' = UNIV :: sym_4 \text{ set}$ (all four cell shapes — BIT0, BIT1, BLANK4, LE4);
- $bl' = BLANK_4, le' = LE_4$;
- $\delta' = \text{alphabet_reduce_delta } M$ (the five-substep union under the global LE-preservation and ar_valid_stage filters);
- $s' = (s_M, \text{ar_init_stage } bl_M), t' = (t_M, \text{ar_accept_stage } bl_M), r' = (r_M, \text{ar_reject_stage } bl_M)$ (M's start / accept / reject states paired with the matching ar_substep_idx tag).

definition $\text{alphabet_reduce} ::$

$('q, 'a) \text{ mttm}$
 $\Rightarrow ('q \times 'a \text{ ar_stage}, sym_4) \text{ mttm}$

where

$\text{alphabet_reduce } M =$
 $(\text{case } M \text{ of } MTTM \ Q_M \ \Gamma_M \ bl_M \ _\ _\ s_M \ t_M \ r_M \ k_M \Rightarrow$
 $MTTM \ (Q_M \times \{\text{stg. ar_valid_stage } \Gamma_M \text{ bl_M stg}$
 $\wedge \text{ar_stage_bounded } bl_M \ k_M \text{ stg}\})$
 $\{\text{BIT0}, \text{BIT1}\}$
 $(UNIV :: sym_4 \text{ set})$
 $BLANK_4$
 LE_4
 $(\text{alphabet_reduce_delta } M)$
 $(s_M, \text{ar_init_stage } bl_M)$
 $(t_M, \text{ar_accept_stage } bl_M)$
 $(r_M, \text{ar_reject_stage } bl_M)$
 $k_M)$

Projection-simp lemmas for the reduced machine: each structural accessor reads straight off the $MTTM$ the combinator builds. Marked $[simp]$ so the language and time proofs never re-derive them via $\text{cases } M$; the accept-state projection in particular is the bridge that makes the simulation engine's terminal state syntactically equal to $t_tm \ (\text{alphabet_reduce } M)$.

lemma $\text{alphabet_reduce_Sigma } [simp]:$

$\text{Sigma_tm } (\text{alphabet_reduce } M) = \{\text{BIT0}, \text{BIT1}\}$
 $\langle \text{proof} \rangle$

```

lemma alphabet_reduce_Gamma [simp]:
   $\Gamma\_tm\ (alphabet\_reduce\ M) = (UNIV :: sym4\ set)$ 
  <proof>

lemma alphabet_reduce_bl [simp]:
   $bl\_tm\ (alphabet\_reduce\ M) = BLANK4$ 
  <proof>

lemma alphabet_reduce_le [simp]:
   $le\_tm\ (alphabet\_reduce\ M) = LE4$ 
  <proof>

lemma alphabet_reduce_start [simp]:
   $s\_tm\ (alphabet\_reduce\ M) = (s\_tm\ M, ar\_init\_stage\ (bl\_tm\ M))$ 
  <proof>

lemma alphabet_reduce_accept [simp]:
   $t\_tm\ (alphabet\_reduce\ M) = (t\_tm\ M, ar\_accept\_stage\ (bl\_tm\ M))$ 
  <proof>

lemma alphabet_reduce_reject [simp]:
   $r\_tm\ (alphabet\_reduce\ M) = (r\_tm\ M, ar\_reject\_stage\ (bl\_tm\ M))$ 
  <proof>

lemma alphabet_reduce_delta [simp]:
   $delta\_tm\ (alphabet\_reduce\ M) = alphabet\_reduce\_delta\ M$ 
  <proof>

end
theory AlphabetReduction_Determinism
  imports AlphabetReduction_Delta
begin

```

2.10 Determinism preservation

The alphabet-reduction combinator preserves determinism: if M is deterministic then so is $alphabet_reduce\ M$. The language, time, and well-formedness theorems assume only well-formedness, so they already cover nondeterministic M ; the result below extends the reduction *up* to deterministic machines, turning the determinism-agnostic construction into a determinism-preserving one.

The argument is structural. $alphabet_reduce_delta\ M$ is the union of the five phase builders ar_delta_read , $ar_delta_compute$, ar_delta_write , $ar_delta_advance$, ar_delta_next , intersected with side conditions. Each builder is single-valued: among the five, only $ar_delta_compute$ consults M 's own transition relation, so it is the sole place M -determinism enters; the other four are single-valued by pure case analysis on the bit-counter, the tape index, and the read cell (ar_delta_next additionally needs $t \neq r$,

which *valid_mttm* supplies). The five builders are keyed to disjoint source substep tags, so two transitions sharing a source land in the same builder; and intersection with the side conditions only shrinks the relation, so single-valuedness of the union transfers to *alphabet_reduce_delta* *M* and hence to the reduced machine.

Each per-builder lemma below splits the *source* tuple into its arms with *elim UnE* and closes every arm against the full target builder: with the source fixed the arm discriminants are pinned, so the target collapses to one matching arm. The *block_width_pos* fact ($1 \leq b$) rules out the degenerate $b = 0$ aliasing where the per-bit arm boundaries would coincide. Throughout, *b* abbreviates *block_width* Γ (the per-symbol cell width).

lemma *ar_delta_read_functional*:
assumes *h1*: $(s, a, s1, a1, d1) \in \text{ar_delta_read } M$
and *h2*: $(s, a, s2, a2, d2) \in \text{ar_delta_read } M$
shows $s1 = s2 \wedge a1 = a2 \wedge d1 = d2$
<proof>

lemma *ar_delta_write_functional*:
assumes *h1*: $(s, a, s1, a1, d1) \in \text{ar_delta_write } M$
and *h2*: $(s, a, s2, a2, d2) \in \text{ar_delta_write } M$
shows $s1 = s2 \wedge a1 = a2 \wedge d1 = d2$
<proof>

lemma *ar_delta_advance_functional*:
assumes *h1*: $(s, a, s1, a1, d1) \in \text{ar_delta_advance } M$
and *h2*: $(s, a, s2, a2, d2) \in \text{ar_delta_advance } M$
shows $s1 = s2 \wedge a1 = a2 \wedge d1 = d2$
<proof>

lemma *ar_delta_next_functional*:
assumes *tr*: $t_tm\ M \neq r_tm\ M$
and *h1*: $(s, a, s1, a1, d1) \in \text{ar_delta_next } M$
and *h2*: $(s, a, s2, a2, d2) \in \text{ar_delta_next } M$
shows $s1 = s2 \wedge a1 = a2 \wedge d1 = d2$
<proof>

The compute substep is the one place *M*'s transition relation enters *alphabet_reduce_delta*: a single δ' -tuple per *M*- δ -tuple, matching the source-stage decoded symbol vector *buf*. Single-valuedness here is exactly *M*-determinism transported through that match.

lemma *ar_delta_compute_functional*:
fixes *M* :: $(q, 'a)\ mttm$
assumes *det*: *det_mttm* *M*
and *h1*: $(s, a, s1, a1, d1) \in \text{ar_delta_compute } M$
and *h2*: $(s, a, s2, a2, d2) \in \text{ar_delta_compute } M$
shows $s1 = s2 \wedge a1 = a2 \wedge d1 = d2$
<proof>

Dispatch on the source substep tag: the five builders have pairwise-disjoint source tags, so two transitions out of a common source s are governed by the same builder, and per-builder single-valuedness applies. The two halt tags carry no transition.

lemma *alphabet_reduce_delta_functional*:
fixes $M :: ('q, 'a) \text{ mttm}$
assumes $\text{val}M: \text{valid_mttm } M$
and $\text{det}M: \text{det_mttm } M$
and $h1: (s, a, s1, a1, d1) \in \text{alphabet_reduce_delta } M$
and $h2: (s, a, s2, a2, d2) \in \text{alphabet_reduce_delta } M$
shows $s1 = s2 \wedge a1 = a2 \wedge d1 = d2$
<proof>

Headline result: the alphabet-reduction combinator carries determinism of M over to $\text{alphabet_reduce } M$. With the determinism-agnostic language / time / output theorems this completes the picture for deterministic machines.

theorem *alphabet_reduce_det*:
fixes $M :: ('q, 'a) \text{ mttm}$
assumes $\text{val}M: \text{valid_mttm } M$
and $\text{det}M: \text{det_mttm } M$
shows $\text{det_mttm } (\text{alphabet_reduce } M :: ('q \times 'a \text{ ar_stage}, \text{sym4}) \text{ mttm})$
<proof>

end
theory *AlphabetReduction_Simulation*
imports *AlphabetReduction_Determinism*
begin

2.11 Simulation correspondence

Position correspondence $\text{sim_pos } b$, the AR analogue of AE's ae_decode_pos read forwards (M-position to M'-position). M -position 0 (the mandatory LE cell) maps to M' -position 0 ; M -position $p \geq 1$ maps to the start of its b -cell block at $(p - 1) \cdot b + 1$. The 1-cell LE / b -cell proper layout is intrinsic to the substrate's mandatory single LE cell at position 0 . Throughout, b abbreviates $\text{block_width } \Gamma$ (the per-symbol cell width).

definition $\text{sim_pos} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
 $\text{sim_pos } k \ p = (\text{if } p = 0 \text{ then } 0 \text{ else } (p - 1) * k + 1)$

The proper-region tiling fact: the encoded cells are b -wide, aligned, and disjoint, so a position $\text{sim_pos } b \ p + j$ ($j < b$) lands in another cell's block $[\text{sim_pos } b \ q, \text{sim_pos } b \ q + b)$ exactly when $p = q$. This is the bridge the forward-step tape re-establishment (tcrr_1) and the advance back-walk (notLE) both turn on: a write at M -position q touches the p -block iff $p = q$. Proved by the alignment argument — a block index off by one shifts the position by a full b , past the $j < b$ offset.

lemma *sim_pos_in_block_iff*:

fixes $K\ p\ q\ j :: \text{nat}$
assumes $K1: 1 \leq K$ **and** $p1: 1 \leq p$ **and** $q1: 1 \leq q$ **and** $jK: j < K$
shows $(\text{sim_pos } K\ q \leq \text{sim_pos } K\ p + j \wedge \text{sim_pos } K\ p + j < \text{sim_pos } K\ q + K)$
 $\longleftrightarrow p = q$
 $\langle \text{proof} \rangle$

Per-cell encoding image $\text{cell_repr } \Gamma\ bl\ x$: the b -cell sym4 block that a single source cell of value x occupies on M 's tape (at proper positions $p \geq 1$). The blank bl maps to b consecutive $BLANK4$ cells; every other source symbol maps to its encode_symbol bit-block. Both branches have length b , so the block is uniformly b -wide. The endmarker le is not a case here: le occurs only at position 0 (a 1-cell $LE4$), handled directly in $\text{ar_tape_correspondence}$.

definition $\text{cell_repr} :: 'a\ \text{set} \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{sym4}\ \text{list}$ **where**

$\text{cell_repr } \Gamma\ bl\ x =$
 (if $x = bl$ then $\text{replicate } (\text{block_width } \Gamma)\ BLANK4$
 else $\text{encode_symbol } \Gamma\ bl\ x$)

Read-phase decode correctness for a whole cell block: folding the accumulator from $\text{gamma_unenum } 0$ over $\text{cell_repr } \Gamma\ bl\ x$ recovers x , uniformly across the blank and proper branches. The blank branch is $\text{foldl_ar_acc_blank}$ (seed rewritten to bl via gamma_unenum_zero); the proper branch is $\text{foldl_ar_acc_encode_symbol}$. This is the fact the single-tape read composition discharges its buf tk -correctness against.

lemma $\text{foldl_ar_acc_cell_repr}$:

assumes $\text{finite } \Gamma$ **and** $bl \in \Gamma$ **and** $x \in \Gamma$
shows $\text{foldl } (\text{ar_acc } \Gamma\ bl)\ (\text{gamma_unenum } \Gamma\ bl\ 0)$
 $(\text{cell_repr } \Gamma\ bl\ x) = x$

$\langle \text{proof} \rangle$

Tape-content correspondence under the encoding, the AR analogue of AE's $\text{ae_tape_correspondence}$. M 's tape cell 0 holds le and M' 's cell 0 holds $LE4$; for every proper position $p \geq 1$, the b -cell block on M' starting at $\text{sim_pos } b\ p$ spells out $\text{cell_repr } \Gamma\ bl\ (tM\ p)$. Since M' 's alphabet is the whole of sym4 , no separate gamma-block invariant is carried (AE's $\text{ae_tape_in_gamma_block}$ would be vacuous here): the correspondence pins every M' -cell.

definition $\text{ar_tape_correspondence} ::$

$'a\ \text{set} \Rightarrow 'a \Rightarrow 'a \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow \text{sym4}) \Rightarrow \text{bool}$ **where**
 $\text{ar_tape_correspondence } \Gamma\ le\ bl\ tM\ tM' \longleftrightarrow$
 $tM\ 0 = le \wedge tM'\ 0 = LE4 \wedge$
 $(\forall p. 1 \leq p \longrightarrow$
 $(\forall j. j < \text{block_width } \Gamma \longrightarrow$
 $tM'\ (\text{sim_pos } (\text{block_width } \Gamma)\ p + j) = \text{cell_repr } \Gamma\ bl\ (tM\ p) ! j))$

The simulation relation, stage-granular, the AR analogue of AE's ae_simulates . Holds at AR_SimRead boundaries (M-step start, M-state

not halted) or at the two halt configurations between an M -configuration cM and an M' -configuration cM' :

- the $ar_substep_idx$ tag is $AR_SimRead$ with the embedded M -state not yet in $\{t, r\}$, or the M -state is t / r and the stage is the matching $ar_accept_stage / ar_reject_stage$ (AR's dedicated halt stages, unlike AE's parked $init_stage$);
- the embedded M -state matches M 's state;
- every tape corresponds cell-for-cell via $ar_tape_correspondence$;
- at an $AR_SimRead$ boundary, every head sits at sim_pos b of M 's head.

definition $ar_simulates ::$

$$\begin{aligned}
& ('q, 'a) mttm \\
& \Rightarrow ('a, 'q) mt_config \\
& \Rightarrow (sym4, 'q \times 'a ar_stage) mt_config \\
& \Rightarrow bool \textbf{ where} \\
& ar_simulates\ M\ cM\ cM' \longleftrightarrow \\
& \quad (let\ qM = mt_state\ cM;\ tsM = mt_tape\ cM;\ nM = mt_pos\ cM;\ \\
& \quad \quad full = mt_state\ cM'; \\
& \quad \quad tsM' = mt_tape\ cM';\ nM' = mt_pos\ cM'\ in \\
& \quad (case\ full\ of\ (qM', stg) \Rightarrow \\
& \quad \quad (case\ stg\ of\ (idx, _, _, _, _, _) \Rightarrow \\
& \quad \quad \quad ((idx = AR_SimRead \wedge qM' \notin \{t_tm\ M, r_tm\ M\}) \\
& \quad \quad \quad \vee (qM' = t_tm\ M \wedge stg = ar_accept_stage\ (bl_tm\ M)) \\
& \quad \quad \quad \vee (qM' = r_tm\ M \wedge stg = ar_reject_stage\ (bl_tm\ M))) \\
& \quad \quad \wedge qM = qM' \\
& \quad \quad \wedge (\forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M) \\
& \quad \quad (bl_tm\ M) \\
& \quad \quad \quad (tsM\ k)\ (tsM'\ k)) \\
& \quad \quad \wedge (idx = AR_SimRead \\
& \quad \quad \quad \longrightarrow (\forall k. nM'\ k = sim_pos\ (block_width\ (\Gamma_tm\ M))\ (nM\ k))))))
\end{aligned}$$

Companion invariant carried alongside $ar_simulates$ (the AR analogue of AE's $ae_buffer_in_gamma_block$): at an $AR_SimRead$ boundary the per-tape position-kind flag $posk$ agrees with M 's head being on the left-end marker, $posk\ k = AR_AtLE \longleftrightarrow nM\ k = 0$. This is the bit the read's LE-vs-proper dispatch consumes; the $AR_AtFirstProper / AR_AtFurtherProper$ split is deliberately not pinned (the read re-derives it via look-back). Kept out of $ar_simulates$ proper so the reverse-direction language proof, which shares $ar_simulates$, carries no $posk$ reasoning. Vacuous off the $AR_SimRead$ boundary (terminal accept/reject configs), where the flag is never read again. Sound by the look-back re-sync argument: established at the initial config (all heads at 0, all flags AR_AtLE) and preserved each M -step by $ar_simulates_forward_step$.

definition *ar_posk_consistent* ::
 ('q, 'a) mttm
 $\Rightarrow$ ('a, 'q) mt_config
 $\Rightarrow$ (sym4, 'q $\times$ 'a ar_stage) mt_config $\Rightarrow$ bool **where**
ar_posk_consistent M cM cM' $\longleftrightarrow$
 (case mt_state cM' of (qM', stg) $\Rightarrow$
 (case stg of (idx, _, _, _, posk) $\Rightarrow$
 idx = AR_SimRead
 $\longrightarrow$ ($\forall k. posk\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0$)))

Second companion invariant, the boundary-shape companion (the buffer-in- Γ companion, plus the canonical-position pin). *ar_simulates*'s *AR_SimRead* arm pins only *idx* = *AR_SimRead* — not the current-tape / bit-counter fields, nor that *buf* is a valid alphabet vector — but the read phase (*ar_read_phase*) starts a fresh per-*M*-step scan at tape 0, counter 0, and *ar_valid_stage* requires *buf* in $\Gamma \cup \{bl\}$. So this companion asserts, at an *AR_SimRead* boundary, the canonical entry shape $tk = 0, i = 0, \forall k. buf\ k \in \Gamma \cup \{bl\}$. Vacuous off the boundary (the halt configs), where the read phase never runs. Under value-level tape count it carries two further invariants the reverse walker's read-phase lift needs: the boundary stage is *ar_stage_bounded* (its *buf* / *dvec* / *posk* tails freeze beyond $k_tm\ M$), and the whole config is blank-tailed ($\forall j \geq k_tm\ M. mt_tape\ cM'\ j\ (mt_pos\ cM'\ j) = BLANK4$) — the *src0_b* / *pad0_b* facts the substrate's *alphabet_reduce_delta* support filter forces on every produced step. Established at the initial config (*ar_init_stage*: tape 0, counter 0, *buf* = $\lambda_. bl$) and preserved each *M*-step by the next handshake's non-terminal arm (resets tape / counter to 0; *buf* carries the just-written symbols, all in Γ). Kept separate from *ar_simulates* for the same reason as *ar_posk_consistent* — the reverse-direction language proof shares *ar_simulates* and should carry no forward-only boundary bookkeeping.

definition *ar_at_read_boundary* ::
 ('q, 'a) mttm
 $\Rightarrow$ (sym4, 'q $\times$ 'a ar_stage) mt_config $\Rightarrow$ bool **where**
ar_at_read_boundary M cM' $\longleftrightarrow$
 (case mt_state cM' of (qM', stg) $\Rightarrow$
 (case stg of (idx, tk, i, buf, dvec, posk) $\Rightarrow$
 idx = AR_SimRead
 $\longrightarrow$ ($tk = 0 \wedge i = 0$
 $\wedge (\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\})$
 $\wedge ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(idx, tk, i, buf, dvec, posk)$)))
 $\wedge (\forall j \geq k_tm\ M. mt_tape\ cM'\ j\ (mt_pos\ cM'\ j) = BLANK4)$

end

theory *AlphabetReduction_ForwardSubsteps*
imports *AlphabetReduction_Simulation*
begin

2.12 Per-substep simulation steps

The compute substep: one M' -step from an $AR_SimCompute$ stage whose buf field matches an M - δ -tuple's read vector fires that tuple, landing at the $AR_SimWrite$ stage with M 's post-step state q' , write vector m_a' , and direction vector m_d threaded into the stage. No tape cell changes and no head moves (the substrate write is the read symbol back, direction N); the per-tape $posk$ carries through. The two global δLE filters are discharged reflexively (write equals read, move N); the target-stage validity rests on $valid_mttm$'s δ -range typing ($m_a' k \in \Gamma_tm M$). Throughout, b abbreviates $block_width \Gamma$ (the per-symbol cell width).

lemma $ar_compute_step$:
fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a ar_stage) mt_config$
assumes $vM: valid_mttm M$
and $stg: mt_state c' = (q, AR_SimCompute, tk, i, buf, dvec, posk)$
and $mdelta: (q, buf, q', m_a', m_d) \in delta_tm M$
and $vsrc: ar_valid_stage (\Gamma_tm M) (bl_tm M)$
 $(AR_SimCompute, tk, i, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm M. mt_tape c' j (mt_pos c' j) = BLANK4$
and $src_bounded: ar_stage_bounded (bl_tm M) (k_tm M)$
 $(AR_SimCompute, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step (alphabet_reduce_delta M)$
 $\wedge mt_state c'' = (q', AR_SimWrite, 0, 0, m_a', m_d, posk)$
 $\wedge mt_tape c'' = mt_tape c'$
 $\wedge mt_pos c'' = mt_pos c'$

$\langle proof \rangle$

Displacement bound: an $AR_SimAdvance$ walk on one tape is at most $2b$ cells (the L -from- $AR_AtFurtherProper$ worst case). The $0 < b$ hypothesis is needed: the N -from- AR_AtLE displacement is the constant 1 , which exceeds $2b = 0$. Used to discharge target-stage validity ($Suc i < 2 * b$) for the walk and boundary arms, whose reached bit-counter is bounded by the displacement.

lemma $ar_disp_le_2k$:
assumes $0 < k$
shows $ar_disp k d pk \leq 2 * k$
 $\langle proof \rangle$

The advance walk substep: from an $AR_SimAdvance$ stage with the bit-counter strictly below the per-tape displacement, one M' -step moves the active tape tk 's head one cell L (N elsewhere) and stays in $AR_SimAdvance$ at $Suc i$; the tape contents and all other heads are unchanged. The $a tk \neq LE4$ hypothesis is load-bearing: it both selects this arm (δ 's stepping arm forbids an L -walk off $LE4$) and discharges the backward- δLE filter, which would otherwise reject the L -move on a tape reading $LE4$. Target-stage validity rests on the displacement bound $ar_disp_le_2k$.

lemma *ar_advance_walk_step*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{notLE}: \text{mt_tape } c' \ tk \ (\text{mt_pos } c' \ tk) \neq \text{LE4}$
and $\text{step_lt}: \text{Suc } i < \text{ar_disp } (\text{block_width } (\Gamma_tm \ M)) \ (\text{dvec } tk) \ (\text{posk } tk)$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(\text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(\text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimAdvance}, tk, \text{Suc } i, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c') (tk := \text{mt_pos } c' \ tk - 1)$
 $\langle \text{proof} \rangle$

The advance boundary substep, non-last tape: the final M' -step of tape tk 's walk, handing off to the next tape $k_succ \ tk$ (bit-counter reset to 0, position-kind updated by ar_newpos , other tapes' posk preserved). A single hypothesis covers both firing sub-cases via the arm's own disjunction: the R -sub-case ($\text{dvec } tk = R$, zero displacement, no head move) and the non- R sub-case ($\text{dvec } tk \neq R$, the read not LE4 , counter at $\text{Suc } i = \text{displacement}$, one last L -move). The head conclusion is therefore conditional on $\text{dvec } tk$.

lemma *ar_advance_boundary_step*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{notlast}: \neg \text{is_last_k } M \ tk$
and $\text{fire}: (\text{dvec } tk = \text{dir}.R \wedge i = 0)$
 $\vee (\text{dvec } tk \neq \text{dir}.R$
 $\wedge \text{mt_tape } c' \ tk \ (\text{mt_pos } c' \ tk) \neq \text{LE4}$
 $\wedge \text{Suc } i = \text{ar_disp } (\text{block_width } (\Gamma_tm \ M)) \ (\text{dvec } tk) \ (\text{posk } tk))$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(\text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(\text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimAdvance}, k_succ \ tk, 0, \text{buf}, \text{dvec},$
 $\text{posk}(tk := \text{ar_newpos } (\text{dvec } tk) \ (\text{posk } tk)))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{if } \text{dvec } tk = \text{dir}.R \text{ then } \text{mt_pos } c'$
 $\text{else } (\text{mt_pos } c') (tk := \text{mt_pos } c' \ tk - 1))$
 $\langle \text{proof} \rangle$

The advance boundary substep, last tape: as *ar_advance_boundary_step* but *tk* is the last tape in the enumeration, so the hand-off goes to *AR_SimNext* (with the current-tape field reset to *k_unidx 0*) instead of advancing to *k_succ tk*. Same two firing sub-cases and the same conditional head conclusion; the arm-selection move is the single rule *UnI2* (the third, rightmost arm of *ar_delta_advance*).

lemma *ar_advance_finish_step*:

fixes *M* :: ('q, 'a) *mttm*
and *c'* :: (sym4, 'q × 'a *ar_stage*) *mt_config*
assumes *vM*: *valid_mttm M*
and *stg*: *mt_state c' = (q, AR_SimAdvance, tk, i, buf, dvec, posk)*
and *qQ*: *q ∈ Q_tm M*
and *last*: *is_last_k M tk*
and *fire*: (*dvec tk = dir.R* ∧ *i = 0*)
 ∨ (*dvec tk ≠ dir.R*
 ∧ *mt_tape c' tk (mt_pos c' tk) ≠ LE4*
 ∧ *Suc i = ar_disp (block_width (Γ_tm M)) (dvec tk) (posk tk)*)
and *vsrc*: *ar_valid_stage (Γ_tm M) (bl_tm M)*
 (*AR_SimAdvance, tk, i, buf, dvec, posk*)
and *pad_blank*: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and *src_bounded*: *ar_stage_bounded (bl_tm M) (k_tm M)*
 (*AR_SimAdvance, tk, i, buf, dvec, posk*)
shows $\exists c''. (c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
 ∧ *mt_state c'' = (q, AR_SimNext, k_unidx 0, 0, buf, dvec,*
 posk(tk := ar_neupos (dvec tk) (posk tk)))
 ∧ *mt_tape c'' = mt_tape c'*
 ∧ *mt_pos c'' = (if dvec tk = dir.R then mt_pos c'*
 else (mt_pos c')(tk := mt_pos c' tk - 1))

<proof>

The write LE-skip substep, non-last tape: when *buf tk* is the left-end marker, the cell at *sim_pos 0 = 0* is already *LE4* and need not be rewritten, so the per-tape write phase is skipped — a single all-*N* substep handing off to the next tape *k_succ tk* with the bit-counter still 0. Tape and heads unchanged (the compute-step shape).

lemma *ar_write_le_step*:

fixes *M* :: ('q, 'a) *mttm*
and *c'* :: (sym4, 'q × 'a *ar_stage*) *mt_config*
assumes *vM*: *valid_mttm M*
and *stg*: *mt_state c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)*
and *qQ*: *q ∈ Q_tm M*
and *poskLE*: *posk tk = AR_AtLE*
and *notlast*: $\neg is_last_k\ M\ tk$
and *vsrc*: *ar_valid_stage (Γ_tm M) (bl_tm M)*
 (*AR_SimWrite, tk, 0, buf, dvec, posk*)
and *pad_blank*: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and *src_bounded*: *ar_stage_bounded (bl_tm M) (k_tm M)*
 (*AR_SimWrite, tk, 0, buf, dvec, posk*)

shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimWrite}, k_succ\ tk, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = \text{mt_pos } c'$

$\langle \text{proof} \rangle$

The write LE-skip substep, last tape: as ar_write_le_step but tk is the last tape, so the hand-off goes to AR_SimAdvance (current-tape field reset to $k_unidx\ 0$). Arm 2 of the six-arm ar_delta_write union.

lemma $\text{ar_write_le_finish_step}$:

fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm\ M$
and $\text{poskLE}: \text{posk } tk = \text{AR_AtLE}$
and $\text{last}: \text{is_last_k } M\ tk$
and $\text{vsrc}: \text{ar_valid_stage } (\Gamma_tm\ M) (bl_tm\ M)$
 $(\text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm\ M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (bl_tm\ M) (k_tm\ M)$
 $(\text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimAdvance}, k_unidx\ 0, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = \text{mt_pos } c'$

$\langle \text{proof} \rangle$

The write back-walk substep (proper region, $\text{Suc } i \leq b$): with $\text{buf } tk$ a proper symbol, the head first walks L back across the b -cell block before the forward-write phase; one L -move on tk (N elsewhere), no writes, staying in AR_SimWrite at $\text{Suc } i$. Structurally the $\text{ar_advance_walk_step}$ shape; arm 3 of the union. As there, a $tk \neq \text{LE4}$ selects the arm and discharges the backward- δLE filter.

lemma $\text{ar_write_walk_step}$:

fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm\ M$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{notLE}: \text{mt_tape } c' tk (\text{mt_pos } c' tk) \neq \text{LE4}$
and $\text{step_le}: \text{Suc } i \leq \text{block_width } (\Gamma_tm\ M)$
and $\text{vsrc}: \text{ar_valid_stage } (\Gamma_tm\ M) (bl_tm\ M)$
 $(\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm\ M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (bl_tm\ M) (k_tm\ M)$
 $(\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$

$$\begin{aligned}
& \wedge mt_state\ c'' = (q, AR_SimWrite, tk, Suc\ i, buf, dvec, posk) \\
& \wedge mt_tape\ c'' = mt_tape\ c' \\
& \wedge mt_pos\ c'' = (mt_pos\ c')(tk := mt_pos\ c'\ tk - 1)
\end{aligned}$$

<proof>

The forward-write image is never the left-end marker: it is *BLANK4* (blank symbol) or a bit cell of *encode_symbol* (*BIT0*/*BIT1* only, by *encode_symbol_cell_domain*), in range $j < b$. This is what makes the forward-write arms legal under both δLE filters.

lemma *write_bit_not_LE4*:
assumes $j < block_width\ \Gamma$
shows $write_bit\ \Gamma\ bl\ x\ j \neq LE4$
<proof>

The j -th cell of a block's *cell_repr* is the j -th *write_bit* image ($j < b$), uniformly across the blank and proper branches. This is the bridge from the tape-correspondence cell value (*cell_repr* ... ! j) to the *write_bit* image that the write phase produces, and to the $\neq LE4$ fact the back-walk / bit-write guards need.

lemma *cell_repr_nth_write_bit*:
assumes $j < block_width\ \Gamma$
shows $cell_repr\ \Gamma\ bl\ x\ !\ j = write_bit\ \Gamma\ bl\ x\ j$
<proof>

lemma *cell_repr_nth_not_LE4*:
assumes $j < block_width\ \Gamma$
shows $cell_repr\ \Gamma\ bl\ x\ !\ j \neq LE4$
<proof>

Under the encoding, the only *LE4* cell of the simulated tape is at position 0: every position ≥ 1 lies in some proper block $[sim_pos\ b\ p, sim_pos\ b\ p + b)$ (the *div/mod* decomposition of $pos - 1$) and so carries a *cell_repr* cell, which is never *LE4*. The fact the advance back-walk's *notLE* guard rests on: a head walking through proper cells never reads the left-end marker.

lemma *ar_tape_correspondence_not_LE4*:
assumes *corr*: $ar_tape_correspondence\ \Gamma\ le\ bl\ tM\ tM'$
and $pos1: 1 \leq pos$
shows $tM'\ pos \neq LE4$
<proof>

The forward-write stepping substep (proper region, $b \leq i, Suc\ i < 2b$): the $(i-b)$ -th cell of *cell_repr* (*buf tk*) is written at *tk*, the head moves *R* on *tk* (*N* elsewhere), staying in *AR_SimWrite* at *Suc i*. First per-substep lemma that mutates the tape: the conclusion's *mt_tape* is a nested *fun_upd* writing *write_bit* $\Gamma\ bl\ (buf\ tk)$ $(i-b)$ at *tk*'s head cell. The read precondition $a\ tk \neq LE4$ (proper region is past *LE4*) discharges the backward- δLE filter; *write_bit_not_LE4* discharges the forward filter.

lemma *ar_write_bit_step*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{notLE}: \text{mt_tape } c' \ tk \ (\text{mt_pos } c' \ tk) \neq \text{LE4}$
and $\text{ilo}: \text{block_width } (\Gamma_tm \ M) \leq i$
and $\text{ihi}: \text{Suc } i < 2 * \text{block_width } (\Gamma_tm \ M)$
and $\text{vsrc}: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimWrite}, tk, \text{Suc } i, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = (\text{mt_tape } c') (tk :=$
 $(\text{mt_tape } c' \ tk) (\text{mt_pos } c' \ tk :=$
 $\text{write_bit } (\Gamma_tm \ M) \ (\text{bl_tm } M) \ (\text{buf } tk)$
 $(i - \text{block_width } (\Gamma_tm \ M))))$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c') (tk := \text{Suc } (\text{mt_pos } c' \ tk))$
 $\langle \text{proof} \rangle$

The forward-write boundary substep, non-last tape ($\text{Suc } i = 2b$): the last cell of *cell_repr* ($\text{buf } tk$) is written ($j = i - b = b - 1$), the head moves *R*, and the phase hands off to the next tape $k_succ \ tk$ with the bit-counter reset to 0. The *ar_write_bit_step* tape-change shape; arm 5 of the union.

lemma *ar_write_bit_boundary_step*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{notLE}: \text{mt_tape } c' \ tk \ (\text{mt_pos } c' \ tk) \neq \text{LE4}$
and $\text{notlast}: \neg \text{is_last_k } M \ tk$
and $\text{ihi}: \text{Suc } i = 2 * \text{block_width } (\Gamma_tm \ M)$
and $\text{vsrc}: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimWrite}, k_succ \ tk, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = (\text{mt_tape } c') (tk :=$
 $(\text{mt_tape } c' \ tk) (\text{mt_pos } c' \ tk :=$
 $\text{write_bit } (\Gamma_tm \ M) \ (\text{bl_tm } M) \ (\text{buf } tk)$
 $(i - \text{block_width } (\Gamma_tm \ M))))$

$\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$

$\langle proof \rangle$

The forward-write boundary substep, last tape: as $ar_write_bit_boundary_step$ but tk is the last tape, so after the last-cell write the phase transitions to $AR_SimAdvance$ (current-tape field reset to $k_unidx\ 0$). Arm 6 (rightmost) of the union.

lemma $ar_write_bit_finish_step$:

fixes $M :: ('q, 'a)\ mttm$

and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$

assumes $vM: valid_mttm\ M$

and $stg: mt_state\ c' = (q, AR_SimWrite, tk, i, buf, dvec, posk)$

and $qQ: q \in Q_tm\ M$

and $poskproper: posk\ tk \neq AR_AtLE$

and $notLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$

and $last: is_last_k\ M\ tk$

and $ihl: Suc\ i = 2 * block_width\ (\Gamma_tm\ M)$

and $vsrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$

$(AR_SimWrite, tk, i, buf, dvec, posk)$

and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$

and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimWrite, tk, i, buf, dvec, posk)$

shows $\exists c''. (c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$

$\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$

$\wedge mt_tape\ c'' = (mt_tape\ c')(tk :=$

$(mt_tape\ c'\ tk)(mt_pos\ c'\ tk :=$

$write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$

$(i - block_width\ (\Gamma_tm\ M))))$

$\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$

$\langle proof \rangle$

The read LE substep, non-last tape: at position 0 the head reads $LE4$, sets $buf\ tk := le_tm\ M$ directly (the single LE-cell needs no accumulator), moves R on tk to position 1 , and hands off to the next tape's read. Tape unchanged. Target-stage validity uses $valid_mttm_LE_in_Gamma$ for the $buf\ tk := le_tm\ M$ entry.

lemma $ar_read_le_step$:

fixes $M :: ('q, 'a)\ mttm$

and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$

assumes $vM: valid_mttm\ M$

and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$

and $qQ: q \in Q_tm\ M$

and $notlast: \neg is_last_k\ M\ tk$

and $posk_le: posk\ tk = AR_AtLE$

and $aLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) = LE4$

and $vsrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$

$(AR_SimRead, tk, 0, buf, dvec, posk)$

and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$

and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step (alphabet_reduce_delta M)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, k_succ\ tk, 0,$
 $buf(tk := le_tm\ M), dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$
 $\langle proof \rangle$

The read LE substep, last tape: as $ar_read_le_step$ but tk is the last tape, so the hand-off goes to $AR_SimCompute$ (current-tape field reset to $k_unidx\ 0$), beginning the compute substep. Arm 2 of the seven-arm ar_delta_read union.

lemma $ar_read_le_finish_step$:
fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage) mt_config$
assumes $vM: valid_mttm\ M$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $qQ: q \in Q_tm\ M$
and $last: is_last_k\ M\ tk$
and $posk_le: posk\ tk = AR_AtLE$
and $aLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) = LE4$
and $vsrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step (alphabet_reduce_delta M)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $buf(tk := le_tm\ M), dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$
 $\langle proof \rangle$

The decoder image always lies in $\Gamma \cup \{bl\}$: for $n < card\ \Gamma$ it is $inv_into\ \Gamma\ (gamma_enum\ \Gamma\ bl)\ n$, which is in Γ since $gamma_enum$ is onto $\{..< card\ \Gamma\}$ (bijection); the out-of-range fallback is bl . Totality here means the read arms' buf -update validity holds for any accumulator value without per-arm range reasoning.

lemma $gamma_unenum_mem$:
assumes $finite\ \Gamma$
shows $gamma_unenum\ \Gamma\ bl\ n \in \Gamma \cup \{bl\}$
 $\langle proof \rangle$

The read proper-arm look-back step 1 ($i = 0$): with the position-kind already in the proper region, the head moves L from $sim_pos(p)$ to $sim_pos(p) - 1$ (the cell to inspect for refining the position-kind), no buf change, transitioning to $i = 1$. Tape unchanged; a $tk \neq LE4$ selects the arm and discharges the backward- δLE filter. Arm 3 of the seven-arm union.

lemma *ar_read_lookback1_step*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $\text{notLE}: \text{mt_tape } c' \ tk \ (\text{mt_pos } c' \ tk) \neq \text{LE4}$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, tk, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c') (tk := \text{mt_pos } c' \ tk - 1)$
 $\langle \text{proof} \rangle$

The read proper-arm look-back step 2 ($i = 1$): at $\text{sim_pos}(p) - 1$ the head reads the cell, refines the position-kind (AR_AtFirstProper if that cell is LE4 , i.e. the head was at $\text{sim_pos } 1$, else $\text{AR_AtFurtherProper}$), resets $\text{buf } tk$ to the zero-bits partial decode, and moves R back to $\text{sim_pos}(p)$, transitioning to $i = 2$. The R -move makes δLE trivial even when reading LE4 . Needs $2 \leq b$ for target validity ($i = 2$). Arm 4 of the union.

lemma *ar_read_lookback2_step*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, tk, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(\text{AR_SimRead}, tk, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, tk, \text{Suc } (\text{Suc } 0),$
 $\text{buf}(tk := \text{gamma_unenum } (\Gamma_tm \ M) \ (\text{bl_tm } M) \ 0), \text{dvec},$
 $\text{posk}(tk := \text{if } \text{mt_tape } c' \ tk \ (\text{mt_pos } c' \ tk) = \text{LE4}$
 $\text{then } \text{AR_AtFirstProper} \text{ else } \text{AR_AtFurtherProper}))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c') (tk := \text{Suc } (\text{mt_pos } c' \ tk))$
 $\langle \text{proof} \rangle$

The read proper-arm per-bit stepping substep ($2 \leq i, \text{Suc } i \leq \text{Suc } b$): at $\text{sim_pos}(p) + (i-2)$ the head reads a bit cell and folds it into the par-

tial decode via the $\text{gamma_enum}/\text{gamma_unenum}$ roundtrip $\text{partial}' = 2 \cdot \text{gamma_enum}(\text{buf } tk) + \text{bit_value}(a \text{ } tk)$, moves R , and continues to $i + 1$. Tape unchanged. Needs $2 \leq b$ for target validity at the upper end ($\text{Suc } i = \text{Suc } b$). Arm 5 of the union.

lemma ar_read_bit_step :
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $\text{ilo}: 2 \leq i$
and $\text{ihi}: \text{Suc } i \leq \text{Suc}(\text{block_width } (\Gamma_tm \ M))$
and $\text{kge2}: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{vsrce}: \text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, tk, \text{Suc } i,$
 $\text{buf}(tk := \text{gamma_unenum } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(2 * \text{gamma_enum } (\Gamma_tm \ M) (\text{bl_tm } M) (\text{buf } tk)$
 $+ \text{bit_value } (\text{mt_tape } c' \ tk (\text{mt_pos } c' \ tk))))),$
 $\text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \text{Suc } (\text{mt_pos } c' \ tk))$
 $\langle \text{proof} \rangle$

The read proper-arm per-bit boundary substep ($i = \text{Suc } b$), non-last tape: the last-bit accumulator step (as ar_read_bit_step) leaving $\text{buf } tk$ the fully-decoded M -symbol, then R -move and hand-off to the next tape $k_succ \ tk$ with the bit-counter reset. Arm 6 of the union.

lemma $\text{ar_read_bit_boundary_step}$:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{notlast}: \neg \text{is_last_k } M \ tk$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $\text{ieq}: i = \text{Suc}(\text{block_width } (\Gamma_tm \ M))$
and $\text{vsrce}: \text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$

$\wedge mt_state\ c'' = (q, AR_SimRead, k_succ\ tk, 0,$
 $\quad buf(tk := gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad\quad (2 * gamma_enum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $\quad\quad + bit_value\ (mt_tape\ c'\ tk\ (mt_pos\ c'\ tk))),$
 $\quad\quad dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$
 $\langle proof \rangle$

The read proper-arm per-bit boundary substep, last tape: same last-bit accumulator step, transitioning to *AR_SimCompute* with the current-tape field reset to *k_unidx 0* (all tapes decoded, begin the compute substep). Arm 7 (rightmost) of the union.

lemma *ar_read_bit_finish_step*:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, i, buf, dvec, posk)$
and $qQ: q \in Q_tm\ M$
and $last: is_last_k\ M\ tk$
and $posk_proper: posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and $ieq: i = Suc\ (block_width\ (\Gamma_tm\ M))$
and $vsrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad (AR_SimRead, tk, i, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $\quad (AR_SimRead, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $\quad buf(tk := gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad\quad (2 * gamma_enum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $\quad\quad + bit_value\ (mt_tape\ c'\ tk\ (mt_pos\ c'\ tk))),$
 $\quad\quad dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$
 $\langle proof \rangle$

end
theory *AlphabetReduction_ForwardRead*
imports *AlphabetReduction_ForwardSubsteps*
begin

2.13 Read phase

The accumulator fold stays inside $\Gamma \cup \{bl\}$: on a non-empty bit list the outermost *ar_acc* is a *gamma_unenum* image (always in range by *gamma_unenum_mem*); on the empty list it is the seed. This is the read-loop's *ar_valid_stage* obligation discharged once for the running *buf* value,

independent of how many bits have been folded so far. Throughout, b abbreviates $block_width \ \Gamma$ (the per-symbol cell width).

lemma *foldl_ar_acc_mem*:

assumes *finite* Γ **and** $a \in \Gamma \cup \{bl\}$

shows *foldl* (*ar_acc* Γ *bl*) $a \ xs \in \Gamma \cup \{bl\}$

<proof>

The inner read loop, single tape, proper cell: starting at bit-counter 2 with the partial decode *buf0* and the head at *base* (the first bit cell, *sim_pos* p), the *ar_read_bit_step* arm fires $b - 1$ times, walking R across the first $b - 1$ bit cells and folding each into *buf tk* via *ar_acc*. After the loop the counter is *Suc* b (ready for the boundary substep that reads the b -th, last bit), the head is at $base + (b - 1)$, the tape is untouched, and every other tape's head is where it was. Instantiates *relpow_invariant_chain* with the loop-index-parameterised invariant whose *buf tk* field is the *foldl* (*ar_acc*) *buf0 tk* of the cells read so far — the same *foldl* the read-decode arithmetic (*foldl_ar_acc_encode_symbol*) is stated against, so the per-step buf update is one *foldl_append* rewrite.

Relation-generic read bit-accumulation loop scaffold. The bit-reading leaf (*ar_read_bit_step*) is the *leaf* hypothesis, quantified over config, counter, and buffer (the accumulator mutates each step); union and sub versions are thin instantiations.

lemma *ar_read_bit_loop_gen*:

fixes $M :: ('q, 'a) \text{mttm}$

and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$

and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config rel}$

assumes *leaf*:

$\bigwedge cc \ i \ \text{buf}' . \llbracket \text{valid_mttm } M;$

$\text{mt_state } cc = (q, AR_SimRead, tk, i, \text{buf}', dvec, posk);$

$q \in Q_tm \ M;$

$posk \ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\};$

$2 \leq i;$

$Suc \ i \leq Suc \ (block_width \ (\Gamma_tm \ M));$

$2 \leq block_width \ (\Gamma_tm \ M);$

$ar_valid_stage \ (\Gamma_tm \ M) \ (bl_tm \ M)$

$(AR_SimRead, tk, i, \text{buf}', dvec, posk);$

$\forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = BLANK4;$

$ar_stage_bounded \ (bl_tm \ M) \ (k_tm \ M)$

$(AR_SimRead, tk, i, \text{buf}', dvec, posk) \rrbracket$

$\implies \exists c'' . (cc, c'') \in R'$

$\wedge \text{mt_state } c'' = (q, AR_SimRead, tk, Suc \ i,$

$\text{buf}'(tk := \text{gamma_unenum} \ (\Gamma_tm \ M) \ (bl_tm \ M)$

$(2 * \text{gamma_enum} \ (\Gamma_tm \ M) \ (bl_tm \ M) \ (\text{buf}'$

$tk)$

$+ \text{bit_value} \ (\text{mt_tape } cc \ tk \ (\text{mt_pos } cc \ tk)))) ,$

$dvec, posk)$

$\wedge \text{mt_tape } c'' = \text{mt_tape } cc$

$\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := Suc\ (mt_pos\ cc\ tk))$
and vM : $valid_mttm\ M$
and qQ : $q \in Q_tm\ M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm\ M)$
and $posk_proper$: $posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and stg : $mt_state\ c' = (q, AR_SimRead, tk, 2, buf0, dvec, posk)$
and $buf0_valid$: $\forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and pos_base : $mt_pos\ c'\ tk = base$
and $pad0$: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0$: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 2, buf0, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm\ M)),$
 $buf0(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))\ (buf0\ tk)$
 $(map\ (\lambda m. mt_tape\ c'\ tk\ (base + m))$
 $[0..<block_width\ (\Gamma_tm\ M) - 1])),$
 $dvec, posk)$
 $\wedge mt_pos\ c''\ tk = base + (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c''\ k' = mt_pos\ c'\ k')$
 $\langle proof \rangle$

lemma $ar_read_bit_loop$:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes vM : $valid_mttm\ M$
and qQ : $q \in Q_tm\ M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm\ M)$
and $posk_proper$: $posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and stg : $mt_state\ c' = (q, AR_SimRead, tk, 2, buf0, dvec, posk)$
and $buf0_valid$: $\forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and pos_base : $mt_pos\ c'\ tk = base$
and $pad0$: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0$: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 2, buf0, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (alphabet_reduce_delta\ M))$
 $\wedge (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm\ M)),$
 $buf0(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))\ (buf0\ tk)$
 $(map\ (\lambda m. mt_tape\ c'\ tk\ (base + m))$
 $[0..<block_width\ (\Gamma_tm\ M) - 1])),$
 $dvec, posk)$
 $\wedge mt_pos\ c''\ tk = base + (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c''\ k' = mt_pos\ c'\ k')$
 $\langle proof \rangle$

The proper-cell single-tape read, non-last tape. From an $AR_SimRead$ stage at bit-counter 0 with the head over a proper block $base = sim_pos\ p$ ($p \geq 1$, so $0 < base$) whose current cell is not $LE4$ and $posk\ tk$ proper, the

read fires the two look-back steps, the inner per-bit loop (*ar_read_bit_loop*), and the bit boundary, decoding the *b*-cell block at *base* into *buf tk* and landing at the next tape's *AR_SimRead* (*k_succ tk*, counter *0*). The decoded *buf tk* is the abstract fold *foldl* (*ar_acc* Γ *bl*) (*gamma_unenum* Γ *bl* *0*) over the block's *b* cells – the form the caller collapses to the source symbol via *foldl_ar_acc_cell_repr* once tape-correspondence pins those cells to *cell_repr*. The chain has length *b* + 2 (look-back 1, look-back 2, loop *b* – 1, boundary). The refined *posk tk* records whether the look-back cell was *LE4* (the head sat at *sim_pos 1*).

The snoc step of a left fold over a *map f* [*0..<b*]: for *0* < *b* the fold equals one *ar_acc* applied to the fold over the first *b* – 1 cells and the last cell *f* (*b* – 1). This is the boundary/finish step's last-bit accumulator collapse, shared by both proper-read consumers.

lemma *ar_acc_foldl_upt_last*:

assumes *0* < *k*

shows *foldl* (*ar_acc* Γ *bl*) *a* (*map f* [*0..<k*])

= *ar_acc* Γ *bl*

(*foldl* (*ar_acc* Γ *bl*) *a* (*map f* [*0..<k* – 1])) (*f* (*k* – 1))

<proof>

The proper-cell read prefix: the part of a proper-cell single-tape read shared by the non-last (*boundary*) and last (*finish*) variants. From an *AR_SimRead* stage at bit-counter *0* with the head over a proper block *base* = *sim_pos p* (*0* < *base*) whose cell is not *LE4* and *posk tk* proper, it fires look-back 1, look-back 2, and the inner per-bit loop (*ar_read_bit_loop*), reaching bit-counter *Suc b* with the head at *base* + (*b* – 1) and *buf tk* the fold over the first *b* – 1 cells. The remaining last-bit step (boundary or finish) is added by the consumer. Chain length *Suc b* (*1* + *1* + (*b* – 1)).

Relation-generic read proper-prefix scaffold. The two lookback steps and the bit loop are the *leaf_lb1*, *leaf_lb2*, *leaf_loop* hypotheses; union and sub versions are thin instantiations.

lemma *ar_read_proper_prefix_gen*:

fixes *M* :: ('*q*, '*a*) *mttm*

and *c'* :: (*sym4*, '*q* × '*a* *ar_stage*) *mt_config*

and *R'* :: (*sym4*, '*q* × '*a* *ar_stage*) *mt_config rel*

assumes *leaf_lb1*:

$\bigwedge cc. \llbracket \text{valid_mttm } M;$

mt_state *cc* = (*q*, *AR_SimRead*, *tk*, *0*, *buf*, *dvec*, *posk*);

q ∈ *Q_tm* *M*;

posk tk ∈ {*AR_AtFirstProper*, *AR_AtFurtherProper*};

mt_tape *cc tk* (*mt_pos* *cc tk*) ≠ *LE4*;

ar_valid_stage (Γ_{tm} *M*) (*bl_tm* *M*)

(*AR_SimRead*, *tk*, *0*, *buf*, *dvec*, *posk*);

$\forall j \geq k_{tm} M. \text{mt_tape } cc j (\text{mt_pos } cc j) = \text{BLANK4};$

ar_stage_bounded (*bl_tm* *M*) (*k_tm* *M*)

$(AR_SimRead, tk, 0, buf, dvec, posk) \parallel$
 $\implies \exists c1. (cc, c1) \in R'$
 $\wedge mt_state\ c1 = (q, AR_SimRead, tk, Suc\ 0, buf, dvec, posk)$
 $\wedge mt_tape\ c1 = mt_tape\ cc$
 $\wedge mt_pos\ c1 = (mt_pos\ cc)(tk := mt_pos\ cc\ tk - 1)$
and *leaf_lb2*:
 $\wedge cc. \parallel valid_mttm\ M;$
 $mt_state\ cc = (q, AR_SimRead, tk, Suc\ 0, buf, dvec, posk);$
 $q \in Q_tm\ M;$
 $posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\};$
 $2 \leq block_width\ (\Gamma_tm\ M);$
 $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, Suc\ 0, buf, dvec, posk);$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, Suc\ 0, buf, dvec, posk) \parallel$
 $\implies \exists c2. (cc, c2) \in R'$
 $\wedge mt_state\ c2 = (q, AR_SimRead, tk, Suc\ (Suc\ 0),$
 $buf(tk := gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ 0), dvec,$
 $posk(tk := if\ mt_tape\ cc\ tk\ (mt_pos\ cc\ tk) = LE4$
 $then\ AR_AtFirstProper\ else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c2 = mt_tape\ cc$
 $\wedge mt_pos\ c2 = (mt_pos\ cc)(tk := Suc\ (mt_pos\ cc\ tk))$
and *leaf_loop*:
 $\wedge cc\ buf0'\ posk'\ base'. \parallel valid_mttm\ M; q \in Q_tm\ M; 2 \leq block_width$
 $(\Gamma_tm\ M);$
 $posk'\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\};$
 $mt_state\ cc = (q, AR_SimRead, tk, 2, buf0', dvec, posk');$
 $\forall k. buf0'\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$
 $mt_pos\ cc\ tk = base';$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 2, buf0', dvec, posk') \parallel$
 $\implies \exists c''. (cc, c'') \in R' \wedge (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm$
 $M)),$
 $buf0'(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))\ (buf0'\ tk)$
 $(map\ (\lambda m. mt_tape\ cc\ tk\ (base' + m))$
 $[0..<block_width\ (\Gamma_tm\ M) - 1])),$
 $dvec, posk')$
 $\wedge mt_pos\ c''\ tk = base' + (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge mt_tape\ c'' = mt_tape\ cc$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c''\ k' = mt_pos\ cc\ k')$
and *vM*: $valid_mttm\ M$
and *qQ*: $q \in Q_tm\ M$
and *kge2*: $2 \leq block_width\ (\Gamma_tm\ M)$
and *posk_proper*: $posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and *stg*: $mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and *notLE*: $mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$

and $base_pos: 0 < base$
and $pos_base: mt_pos\ c'\ tk = base$
and $vsrvc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge \wedge Suc\ (block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm\ M)),$
 $buf(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))$
 $(gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ 0)$
 $(map\ (\lambda m. mt_tape\ c'\ tk\ (base + m))$
 $[0..<block_width\ (\Gamma_tm\ M) - 1])),$
 $dvec,$
 $posk(tk := if\ mt_tape\ c'\ tk\ (base - 1) = LE4$
 $then\ AR_AtFirstProper\ else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + (block_width\ (\Gamma_tm\ M) -$
 $1))$
 $\langle proof \rangle$

lemma $ar_read_proper_prefix:$

fixes $M :: ('q, 'a)\ mttm$

and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$

assumes $vM: valid_mttm\ M$

and $qQ: q \in Q_tm\ M$

and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$

and $posk_proper: posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$

and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$

and $notLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$

and $base_pos: 0 < base$

and $pos_base: mt_pos\ c'\ tk = base$

and $vsrvc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$

$(AR_SimRead, tk, 0, buf, dvec, posk)$

and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$

and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimRead, tk, 0, buf, dvec, posk)$

shows $\exists c''. (c', c'') \in (mttm_step\ (alphabet_reduce_delta\ M))$

$\wedge \wedge Suc\ (block_width\ (\Gamma_tm\ M))$

$\wedge mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm\ M)),$

$buf(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))$

$(gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ 0)$

$(map\ (\lambda m. mt_tape\ c'\ tk\ (base + m))$

$[0..<block_width\ (\Gamma_tm\ M) - 1])),$

$dvec,$

$posk(tk := if\ mt_tape\ c'\ tk\ (base - 1) = LE4$

$then\ AR_AtFirstProper\ else\ AR_AtFurtherProper))$

$\wedge mt_tape\ c'' = mt_tape\ c'$

$\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + (block_width\ (\Gamma_tm\ M) -$

1))
 ⟨proof⟩

The proper-cell single-tape read, non-last tape: the prefix followed by the bit-boundary step (*ar_read_bit_boundary_step*), decoding the *b*-cell block at *base* into *buf tk* (the abstract fold *foldl* (*ar_acc* Γ *bl*) (*gamma_unenum* Γ *bl* 0), collapsed to the source symbol by *foldl_ar_acc_cell_repr*) and landing at the next tape's *AR_SimRead* (*k_succ tk*, counter 0). Chain length *b* + 2.

Relation-generic read proper-step scaffold. The prefix walk and the closing bit-boundary step are the *leaf_prefix* and *leaf_boundary* hypotheses; union and sub versions are thin instantiations.

lemma *ar_read_proper_step_gen*:

fixes *M* :: ('q, 'a) mttm

and *c'* :: (sym4, 'q × 'a ar_stage) mt_config

and *R'* :: (sym4, 'q × 'a ar_stage) mt_config rel

assumes *leaf_prefix*:

$\bigwedge cc. \llbracket \text{valid_mttm } M; q \in Q_tm\ M; 2 \leq \text{block_width } (\Gamma_tm\ M);$

$\text{posk } tk \in \{AR_AtFirstProper, AR_AtFurtherProper\};$

$\text{mt_state } cc = (q, AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk});$

$\text{mt_tape } cc\ tk\ (\text{mt_pos } cc\ tk) \neq LE4;$

$0 < \text{base};$

$\text{mt_pos } cc\ tk = \text{base};$

$\text{ar_valid_stage } (\Gamma_tm\ M)\ (\text{bl_tm } M)$

$(AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk});$

$\forall j \geq k_tm\ M. \text{mt_tape } cc\ j\ (\text{mt_pos } cc\ j) = BLANK4;$

$\text{ar_stage_bounded } (\text{bl_tm } M)\ (k_tm\ M)$

$(AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \rrbracket$

$\implies \exists c''. (cc, c'') \in R' \wedge \wedge \text{Suc } (\text{block_width } (\Gamma_tm\ M))$

$\wedge \text{mt_state } c'' = (q, AR_SimRead, tk, \text{Suc } (\text{block_width } (\Gamma_tm$

M)),

$\text{buf}(tk := \text{foldl } (\text{ar_acc } (\Gamma_tm\ M)\ (\text{bl_tm } M))$

$(\text{gamma_unenum } (\Gamma_tm\ M)\ (\text{bl_tm } M)\ 0)$

$(\text{map } (\lambda m. \text{mt_tape } cc\ tk\ (\text{base} + m))$

$[0..<\text{block_width } (\Gamma_tm\ M) - 1]))$,

dvec,

$\text{posk}(tk := \text{if } \text{mt_tape } cc\ tk\ (\text{base} - 1) = LE4$

$\text{then } AR_AtFirstProper \text{ else } AR_AtFurtherProper))$

$\wedge \text{mt_tape } c'' = \text{mt_tape } cc$

$\wedge \text{mt_pos } c'' = (\text{mt_pos } cc)(tk := \text{base} + (\text{block_width } (\Gamma_tm$

M) - 1))

and *leaf_boundary*:

$\bigwedge cc\ i\ \text{buf}'\ \text{posk}'. \llbracket \text{valid_mttm } M;$

$\text{mt_state } cc = (q, AR_SimRead, tk, i, \text{buf}', \text{dvec}, \text{posk}')$

$q \in Q_tm\ M;$

$\neg \text{is_last } k\ M\ tk;$

$\text{posk}'\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\};$

$i = \text{Suc } (\text{block_width } (\Gamma_tm\ M));$

$ar_valid_stage (\Gamma_tm M) (bl_tm M)$
 $(AR_SimRead, tk, i, buf', dvec, posk')$;
 $\forall j \geq k_tm M. mt_tape cc j (mt_pos cc j) = BLANK4$;
 $ar_stage_bounded (bl_tm M) (k_tm M)$
 $(AR_SimRead, tk, i, buf', dvec, posk') \parallel$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge mt_state c'' = (q, AR_SimRead, k_succ tk, 0,$
 $buf'(tk := gamma_unenum (\Gamma_tm M) (bl_tm M)$
 $(2 * gamma_enum (\Gamma_tm M) (bl_tm M) (buf' tk)$
 $+ bit_value (mt_tape cc tk (mt_pos cc tk))))),$
 $dvec, posk')$
 $\wedge mt_tape c'' = mt_tape cc$
 $\wedge mt_pos c'' = (mt_pos cc)(tk := Suc (mt_pos cc tk))$
and vM : $valid_mttm M$
and qQ : $q \in Q_tm M$
and $kge2$: $2 \leq block_width (\Gamma_tm M)$
and $notlast$: $\neg is_last_k M tk$
and $posk_proper$: $posk tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and stg : $mt_state c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $notLE$: $mt_tape c' tk (mt_pos c' tk) \neq LE4$
and $base_pos$: $0 < base$
and pos_base : $mt_pos c' tk = base$
and $vsrsc$: $ar_valid_stage (\Gamma_tm M) (bl_tm M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0$: $\forall j \geq k_tm M. mt_tape c' j (mt_pos c' j) = BLANK4$
and $src0$: $ar_stage_bounded (bl_tm M) (k_tm M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge (block_width (\Gamma_tm M) + 2)$
 $\wedge mt_state c'' = (q, AR_SimRead, k_succ tk, 0,$
 $buf(tk := foldl (ar_acc (\Gamma_tm M) (bl_tm M))$
 $(gamma_unenum (\Gamma_tm M) (bl_tm M) 0)$
 $(map (\lambda m. mt_tape c' tk (base + m))$
 $[0..<block_width (\Gamma_tm M)])),$
 $dvec,$
 $posk(tk := if mt_tape c' tk (base - 1) = LE4$
 $then AR_AtFirstProper else AR_AtFurtherProper))$
 $\wedge mt_tape c'' = mt_tape c'$
 $\wedge mt_pos c'' = (mt_pos c')(tk := base + block_width (\Gamma_tm M))$
 $\langle proof \rangle$

lemma $ar_read_proper_step$:

fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a ar_stage) mt_config$
assumes vM : $valid_mttm M$
and qQ : $q \in Q_tm M$
and $kge2$: $2 \leq block_width (\Gamma_tm M)$
and $notlast$: $\neg is_last_k M tk$
and $posk_proper$: $posk tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and stg : $mt_state c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$

```

and notLE: mt_tape c' tk (mt_pos c' tk)  $\neq$  LE4
and base_pos:  $0 < \text{base}$ 
and pos_base: mt_pos c' tk = base
and vsr: ar_valid_stage ( $\Gamma_{tm}$  M) (bl_tm M)
           (AR_SimRead, tk,  $0$ , buf, dvec, posk)
and pad0:  $\forall j \geq k_{tm} M. \text{mt\_tape } c' j (\text{mt\_pos } c' j) = \text{BLANK4}$ 
and src0: ar_stage_bounded (bl_tm M) (k_tm M)
           (AR_SimRead, tk,  $0$ , buf, dvec, posk)
shows  $\exists c''. (c', c'') \in (\text{mttm\_step } (\text{alphabet\_reduce\_delta } M))$ 
            $\wedge (\text{block\_width } (\Gamma_{tm} M) + 2)$ 
            $\wedge \text{mt\_state } c'' = (q, \text{AR\_SimRead}, k_{succ} tk, 0,$ 
            $\text{buf}(tk := \text{foldl } (\text{ar\_acc } (\Gamma_{tm} M) (\text{bl\_tm } M))$ 
            $(\text{gamma\_unenum } (\Gamma_{tm} M) (\text{bl\_tm } M) 0)$ 
            $(\text{map } (\lambda m. \text{mt\_tape } c' tk (\text{base} + m))$ 
            $[0..<\text{block\_width } (\Gamma_{tm} M)]))$ ,
           dvec,
           posk(tk := if mt_tape c' tk (base -  $1$ ) = LE4
           then AR_AtFirstProper else AR_AtFurtherProper))
            $\wedge \text{mt\_tape } c'' = \text{mt\_tape } c'$ 
            $\wedge \text{mt\_pos } c'' = (\text{mt\_pos } c')(tk := \text{base} + \text{block\_width } (\Gamma_{tm} M))$ 
<proof>

```

The proper-cell single-tape read, last tape: the prefix followed by the bit-finish step (*ar_read_bit_finish_step*), decoding the *b*-cell block at *base* into *buf tk* exactly as the non-last variant, but transitioning to *AR_SimCompute* with the current-tape field reset to *k_unidx 0* (all tapes decoded). Chain length *b* + 2.

Relation-generic read proper-finish-step scaffold. As *ar_read_proper_step_gen* but the closing leaf is the last-tape bit-finish (transition to *AR_SimCompute*); the prefix and finish helpers are the *leaf_prefix* and *leaf_finish* hypotheses.

lemma *ar_read_proper_finish_step_gen*:

fixes *M* :: ('q, 'a) *mttm*

and *c'* :: (*sym4*, 'q $\times$ 'a *ar_stage*) *mt_config*

and *R'* :: (*sym4*, 'q $\times$ 'a *ar_stage*) *mt_config rel*

assumes *leaf_prefix*:

```

 $\wedge cc. \llbracket \text{valid\_mttm } M; q \in Q_{tm} M; 2 \leq \text{block\_width } (\Gamma_{tm} M);$ 
           posk tk  $\in \{\text{AR\_AtFirstProper}, \text{AR\_AtFurtherProper}\};$ 
           mt_state cc = (q, AR_SimRead, tk,  $0$ , buf, dvec, posk);
           mt_tape cc tk (mt_pos cc tk)  $\neq$  LE4;
            $0 < \text{base};$ 
           mt_pos cc tk = base;
           ar_valid_stage ( $\Gamma_{tm}$  M) (bl_tm M)
           (AR_SimRead, tk,  $0$ , buf, dvec, posk);
            $\forall j \geq k_{tm} M. \text{mt\_tape } cc j (\text{mt\_pos } cc j) = \text{BLANK4};$ 
           ar_stage_bounded (bl_tm M) (k_tm M)
           (AR_SimRead, tk,  $0$ , buf, dvec, posk)  $\rrbracket$ 
 $\implies \exists c''. (cc, c'') \in R' \wedge \text{Suc } (\text{block\_width } (\Gamma_{tm} M))$ 

```

$\wedge mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm\ M))),$
 $buf(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))$
 $(gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ 0)$
 $(map\ (\lambda m. mt_tape\ cc\ tk\ (base + m))$
 $[0..<block_width\ (\Gamma_tm\ M) - 1])),$
 $dvec,$
 $posk(tk := if\ mt_tape\ cc\ tk\ (base - 1) = LE4$
 $then\ AR_AtFirstProper\ else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ cc$
 $\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := base + (block_width\ (\Gamma_tm\ M) - 1))$
and *leaf_finish*:
 $\wedge cc\ i\ buf'\ posk'. \llbracket valid\ mttm\ M;$
 $mt_state\ cc = (q, AR_SimRead, tk, i, buf', dvec, posk');$
 $q \in Q_tm\ M;$
 $is_last_k\ M\ tk;$
 $posk'\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\};$
 $i = Suc\ (block_width\ (\Gamma_tm\ M));$
 $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, i, buf', dvec, posk');$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, i, buf', dvec, posk') \rrbracket$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $buf'(tk := gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(2 * gamma_enum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf'\ tk)$
 $+ bit_value\ (mt_tape\ cc\ tk\ (mt_pos\ cc\ tk)))))$
 $dvec, posk')$
 $\wedge mt_tape\ c'' = mt_tape\ cc$
 $\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := Suc\ (mt_pos\ cc\ tk))$
and *vM*: *valid_mttm M*
and *qQ*: $q \in Q_tm\ M$
and *kge2*: $2 \leq block_width\ (\Gamma_tm\ M)$
and *last*: *is_last_k M tk*
and *posk_proper*: $posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and *stg*: $mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and *notLE*: $mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$
and *base_pos*: $0 < base$
and *pos_base*: $mt_pos\ c'\ tk = base$
and *usrc*: $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and *pad0*: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and *src0*: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $buf(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))$

$(\text{gamma_unenum } (\Gamma_tm\ M) (bl_tm\ M)\ 0)$
 $(\text{map } (\lambda m. mt_tape\ c'\ tk\ (base + m))$
 $\quad [0..<block_width\ (\Gamma_tm\ M)])),$
 $dvec,$
 $\text{posk}(tk := \text{if } mt_tape\ c'\ tk\ (base - 1) = LE4$
 $\quad \text{then } AR_AtFirstProper\ \text{else } AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + block_width\ (\Gamma_tm\ M))$
 $\langle proof \rangle$

lemma $ar_read_proper_finish_step$:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $last: is_last_k\ M\ tk$
and $posk_proper: posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $notLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$
and $base_pos: 0 < base$
and $pos_base: mt_pos\ c'\ tk = base$
and $vsrsc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad (AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $\quad (AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (alphabet_reduce_delta\ M))$
 $\quad \wedge (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $\quad buf(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))$
 $\quad \quad (\text{gamma_unenum } (\Gamma_tm\ M) (bl_tm\ M)\ 0)$
 $\quad \quad (\text{map } (\lambda m. mt_tape\ c'\ tk\ (base + m))$
 $\quad \quad [0..<block_width\ (\Gamma_tm\ M)])),$
 $dvec,$
 $\text{posk}(tk := \text{if } mt_tape\ c'\ tk\ (base - 1) = LE4$
 $\quad \text{then } AR_AtFirstProper\ \text{else } AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + block_width\ (\Gamma_tm\ M))$
 $\langle proof \rangle$

Unified single-tape read, non-last tape: dispatches the per-tape read on whether M 's head sits on the left-end marker ($p = 0$, the LE arm) or in the proper region ($p \geq 1$, the proper arm) from the per-tape correspondence facts rather than the raw cell pattern. Both arms land back at $AR_SimRead$ on the successor tape with $buf\ tk$ holding the decoded M -symbol $tM\ p$ (via $foldl_ar_acc_cell_repr$) and $posk\ tk$ re-synced exact to the head's position kind (AR_AtLE at 0, $AR_AtFirstProper$ at 1, $AR_AtFurtherProper$ beyond — the marker test $tM'(base - 1) = LE4$ coincides with $p = 1$ by

the left-end discipline). Cost 1 (LE) or $b + 2$ (proper). This is the uniform per-tape step the read-phase tape walk iterates.

Relation-generic read single-tape step scaffold (non-last). The two cases ($p = 0$ le-step, $p \neq 0$ proper-step) are the *leaf_le* and *leaf_proper* hypotheses; union and sub versions are thin instantiations.

lemma *ar_read_tape_step_gen*:

fixes $M :: ('q, 'a) \text{ mttm}$

and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$

and $tM :: \text{nat} \Rightarrow 'a$

and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config rel}$

assumes *leaf_le*:

$\bigwedge cc. \llbracket \text{valid_mttm } M;$
 $\text{mt_state } cc = (q, \text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk});$
 $q \in Q_tm \ M;$
 $\neg \text{is_last_k } M \ tk;$
 $\text{posk } tk = \text{AR_AtLE};$
 $\text{mt_tape } cc \ tk \ (\text{mt_pos } cc \ tk) = \text{LE4};$
 $\text{ar_valid_stage } (\Gamma_tm \ M) \ (bl_tm \ M)$
 $\ (\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk});$
 $\forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (bl_tm \ M) \ (k_tm \ M)$
 $\ (\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, k_succ \ tk, 0,$
 $\text{buf}(tk := \text{le_tm } M), \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } cc$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } cc)(tk := \text{Suc } (\text{mt_pos } cc \ tk))$

and *leaf_proper*:

$\bigwedge cc \text{ base}'. \llbracket \text{valid_mttm } M; q \in Q_tm \ M; 2 \leq \text{block_width } (\Gamma_tm \ M);$
 $\neg \text{is_last_k } M \ tk;$
 $\text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\};$
 $\text{mt_state } cc = (q, \text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk});$
 $\text{mt_tape } cc \ tk \ (\text{mt_pos } cc \ tk) \neq \text{LE4};$
 $0 < \text{base}';$
 $\text{mt_pos } cc \ tk = \text{base}';$
 $\text{ar_valid_stage } (\Gamma_tm \ M) \ (bl_tm \ M)$
 $\ (\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk});$
 $\forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (bl_tm \ M) \ (k_tm \ M)$
 $\ (\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R' \wedge \wedge (\text{block_width } (\Gamma_tm \ M) + 2)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, k_succ \ tk, 0,$
 $\text{buf}(tk := \text{foldl } (\text{ar_acc } (\Gamma_tm \ M) \ (bl_tm \ M))$
 $\ (\text{gamma_unenum } (\Gamma_tm \ M) \ (bl_tm \ M) \ 0)$
 $\ (\text{map } (\lambda m. \text{mt_tape } cc \ tk \ (\text{base}' + m))$
 $\ [0..<\text{block_width } (\Gamma_tm \ M)]),$
 $\text{dvec},$
 $\text{posk}(tk := \text{if } \text{mt_tape } cc \ tk \ (\text{base}' - 1) = \text{LE4}$

$\text{then } AR_AtFirstProper \text{ else } AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ cc$
 $\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := base' + block_width$

$(\Gamma_tm\ M))$
and vM : $valid_mttm\ M$
and qQ : $q \in Q_tm\ M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm\ M)$
and $notlast$: $\neg is_last_k\ M\ tk$
and stg : $mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $tcrr$: $ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $tM\ (mt_tape\ c'\ tk)$
and $ppos$: $mt_pos\ c'\ tk = sim_pos\ (block_width\ (\Gamma_tm\ M))\ p$
and $pkok$: $posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $proper_mem$: $1 \leq p \implies tM\ p \in \Gamma_tm\ M$
and vsr : $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0$: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0$: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge (if\ p = 0\ then\ 1\ else\ block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, k_succ\ tk, 0,$
 $buf(tk := tM\ p), dvec,$
 $posk(tk := if\ p = 0\ then\ AR_AtLE$
 $else\ if\ p = 1\ then\ AR_AtFirstProper$
 $else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk :=$
 $if\ p = 0\ then\ Suc\ 0$
 $else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
 $\langle proof \rangle$

lemma $ar_read_tape_step$:

fixes M :: $('q, 'a)\ mttm$
and c' :: $(sym4, 'q \times 'a\ ar_stage)\ mt_config$
and tM :: $nat \Rightarrow 'a$
assumes vM : $valid_mttm\ M$
and qQ : $q \in Q_tm\ M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm\ M)$
and $notlast$: $\neg is_last_k\ M\ tk$
and stg : $mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $tcrr$: $ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $tM\ (mt_tape\ c'\ tk)$
and $ppos$: $mt_pos\ c'\ tk = sim_pos\ (block_width\ (\Gamma_tm\ M))\ p$
and $pkok$: $posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $proper_mem$: $1 \leq p \implies tM\ p \in \Gamma_tm\ M$
and vsr : $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0$: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$

and $src0$: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (alphabet_reduce_delta\ M))$
 $\wedge (if\ p = 0\ then\ 1\ else\ block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, k_succ\ tk, 0,$
 $buf(tk := tM\ p), dvec,$
 $posk(tk := if\ p = 0\ then\ AR_AtLE$
 $else\ if\ p = 1\ then\ AR_AtFirstProper$
 $else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk :=$
 $if\ p = 0\ then\ Suc\ 0$
 $else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
 $\langle proof \rangle$

Unified single-tape read, last tape: as $ar_read_tape_step$ but tk is the last tape, so both arms transition to $AR_SimCompute$ with the current-tape field reset to $k_unidx\ 0$ (all tapes decoded, the read phase complete). The $buf\ tk$ decode and $posk\ tk$ re-sync are identical to the non-last variant; only the hand-off target differs.

Relation-generic read single-tape finish-step scaffold (last tape). As $ar_read_tape_step_gen$ but both cases use the last-tape finish leaves (transition to $AR_SimCompute$); the le-finish and proper-finish helpers are the $leaf_le$ and $leaf_proper$ hypotheses.

lemma $ar_read_tape_finish_step_gen$:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $tM :: nat \Rightarrow 'a$
and $R' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config\ rel$
assumes $leaf_le$:
 $\wedge cc. \llbracket valid_mttm\ M;$
 $mt_state\ cc = (q, AR_SimRead, tk, 0, buf, dvec, posk);$
 $q \in Q_tm\ M;$
 $is_last_k\ M\ tk;$
 $posk\ tk = AR_AtLE;$
 $mt_tape\ cc\ tk\ (mt_pos\ cc\ tk) = LE4;$
 $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk);$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $buf(tk := le_tm\ M), dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ cc$
 $\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := Suc\ (mt_pos\ cc\ tk))$
and $leaf_proper$:

$$\begin{aligned}
& \wedge cc \text{ base}'. \llbracket \text{valid_mttm } M; q \in Q_tm \ M; 2 \leq \text{block_width } (\Gamma_tm \ M); \\
& \quad \text{is_last_k } M \ tk; \\
& \quad \text{posk } tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}; \\
& \quad \text{mt_state } cc = (q, AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk}); \\
& \quad \text{mt_tape } cc \ tk \ (\text{mt_pos } cc \ tk) \neq LE4; \\
& \quad 0 < \text{base}'; \\
& \quad \text{mt_pos } cc \ tk = \text{base}'; \\
& \quad \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M) \\
& \quad \quad (AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk}); \\
& \quad \forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = BLANK4; \\
& \quad \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M) \\
& \quad \quad (AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \rrbracket \\
& \implies \exists c''. (cc, c'') \in R' \wedge (block_width (\Gamma_tm \ M) + 2) \\
& \quad \wedge \text{mt_state } c'' = (q, AR_SimCompute, k_unidx \ 0, 0, \\
& \quad \quad \text{buf}(tk := \text{foldl } (\text{ar_acc } (\Gamma_tm \ M) \ (\text{bl_tm } M)) \\
& \quad \quad \quad (\text{gamma_unenum } (\Gamma_tm \ M) \ (\text{bl_tm } M) \ 0) \\
& \quad \quad \quad (\text{map } (\lambda m. \text{mt_tape } cc \ tk \ (\text{base}' + m)) \\
& \quad \quad \quad [0..<\text{block_width } (\Gamma_tm \ M)])), \\
& \quad \quad \text{dvec}, \\
& \quad \quad \text{posk}(tk := \text{if } \text{mt_tape } cc \ tk \ (\text{base}' - 1) = LE4 \\
& \quad \quad \quad \text{then } AR_AtFirstProper \text{ else } AR_AtFurtherProper)) \\
& \quad \wedge \text{mt_tape } c'' = \text{mt_tape } cc \\
& \quad \wedge \text{mt_pos } c'' = (\text{mt_pos } cc)(tk := \text{base}' + \text{block_width} \\
& \quad (\Gamma_tm \ M)) \\
& \quad \text{and } vM: \text{valid_mttm } M \\
& \quad \text{and } qQ: q \in Q_tm \ M \\
& \quad \text{and } kge2: 2 \leq \text{block_width } (\Gamma_tm \ M) \\
& \quad \text{and } last: \text{is_last_k } M \ tk \\
& \quad \text{and } stg: \text{mt_state } c' = (q, AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \\
& \quad \text{and } tcorr: \text{ar_tape_correspondence } (\Gamma_tm \ M) \ (le_tm \ M) \ (\text{bl_tm } M) \\
& \quad \quad tM \ (\text{mt_tape } c' \ tk) \\
& \quad \text{and } ppos: \text{mt_pos } c' \ tk = \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) \ p \\
& \quad \text{and } pkok: \text{posk } tk = AR_AtLE \iff p = 0 \\
& \quad \text{and } proper_mem: 1 \leq p \implies tM \ p \in \Gamma_tm \ M \\
& \quad \text{and } vsrc: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M) \\
& \quad \quad (AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \\
& \quad \text{and } pad0: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = BLANK4 \\
& \quad \text{and } src0: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M) \\
& \quad \quad (AR_SimRead, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \\
& \text{shows } \exists c''. (c', c'') \in R' \wedge (\text{if } p = 0 \text{ then } 1 \text{ else } \text{block_width } (\Gamma_tm \ M) + 2) \\
& \quad \wedge \text{mt_state } c'' = (q, AR_SimCompute, k_unidx \ 0, 0, \\
& \quad \quad \text{buf}(tk := tM \ p), \text{dvec}, \\
& \quad \quad \text{posk}(tk := \text{if } p = 0 \text{ then } AR_AtLE \\
& \quad \quad \quad \text{else if } p = 1 \text{ then } AR_AtFirstProper \\
& \quad \quad \quad \text{else } AR_AtFurtherProper)) \\
& \quad \wedge \text{mt_tape } c'' = \text{mt_tape } c' \\
& \quad \wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \\
& \quad \quad \text{if } p = 0 \text{ then } \text{Suc } 0 \\
& \quad \quad \text{else } \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) \ p + \text{block_width } (\Gamma_tm
\end{aligned}$$

$M))$
 $\langle proof \rangle$

lemma *ar_read_tape_finish_step*:
fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a ar_stage) mt_config$
and $tM :: nat \Rightarrow 'a$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $last: is_last_k\ M\ tk$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $tcrr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $tM\ (mt_tape\ c'\ tk)$
and $ppos: mt_pos\ c'\ tk = sim_pos\ (block_width\ (\Gamma_tm\ M))\ p$
and $pkok: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $proper_mem: 1 \leq p \implies tM\ p \in \Gamma_tm\ M$
and $vsrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (alphabet_reduce_delta\ M))$
 $\wedge (if\ p = 0\ then\ 1\ else\ block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $buf(tk := tM\ p), dvec,$
 $posk(tk := if\ p = 0\ then\ AR_AtLE$
 $else\ if\ p = 1\ then\ AR_AtFirstProper$
 $else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk :=$
 $if\ p = 0\ then\ Suc\ 0$
 $else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
 $\langle proof \rangle$

The read-phase prefix walk: starting from the read boundary (current-tape field $k_unidx\ 0$, bit-counter 0), iterate the unified non-last per-tape read *ar_read_tape_step* over the first j tapes of the enumeration ($j \leq k_tm\ M - 1$, so every tape touched is non-last), landing back at *AR_SimRead* on tape $k_unidx\ j$. The walk descriptor splits on $k_idx\ k < j$: tapes already visited carry the decoded M -symbol in *buf*, the exact re-synced *posk*, and the block-end head position; the rest retain their boundary values. Variable per-tape cost (1 or $b + 2$), aggregate bounded by $j \cdot (b + 2)$. Custom induction on j ; the inductive step's function-update bookkeeping rests on k_idx injectivity ($k \neq k_unidx\ j \implies k_idx\ k \neq j$).

Relation-generic read prefix walk scaffold. The single per-tape helper (*ar_read_tape_step*) is the *leaf* hypothesis, quantified over the per-tape

config, tape index, buffer, position-kind, tape-content function, and position;
union and sub versions are thin instantiations.

lemma *ar_read_prefix_gen*:

fixes $M :: ('q, 'a) \text{mttm}$
and $cM :: ('a, 'q) \text{mt_config}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$
and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config rel}$
assumes *leaf*:
 $\bigwedge cc \ tk' \ buf' \ posk' \ tM' \ p'.$
 $\llbracket \text{valid_mttm } M; q \in Q_tm \ M; 2 \leq \text{block_width } (\Gamma_tm \ M);$
 $\neg \text{is_last_k } M \ tk';$
 $\text{mt_state } cc = (q, \text{AR_SimRead}, tk', 0, buf', dvec, posk');$
 $\text{ar_tape_correspondence } (\Gamma_tm \ M) (\text{le_tm } M) (\text{bl_tm } M)$
 $\text{tM'} (mt_tape \ cc \ tk');$
 $\text{mt_pos } cc \ tk' = \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) \ p';$
 $\text{posk'} \ tk' = \text{AR_AtLE} \longleftrightarrow p' = 0;$
 $1 \leq p' \implies tM' \ p' \in \Gamma_tm \ M;$
 $\text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk', 0, buf', dvec, posk');$
 $\forall j \geq k_tm \ M. \text{mt_tape } cc \ j (\text{mt_pos } cc \ j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimRead}, tk', 0, buf', dvec, posk') \rrbracket$
 $\implies \exists c''. (cc, c'') \in R' \wedge (if \ p' = 0 \text{ then } 1 \text{ else } \text{block_width } (\Gamma_tm \ M))$
 $+ \ 2)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, k_succ \ tk', 0,$
 $\text{buf'}(tk' := tM' \ p'), dvec,$
 $\text{posk'}(tk' := if \ p' = 0 \text{ then } \text{AR_AtLE}$
 $\text{else if } p' = 1 \text{ then } \text{AR_AtFirstProper}$
 $\text{else } \text{AR_AtFurtherProper}))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } cc$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } cc)(tk' :=$
 $if \ p' = 0 \text{ then } \text{Suc } 0$
 $\text{else } \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) \ p' + \text{block_width}$
 $(\Gamma_tm \ M))$
and $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $stg0: \text{mt_state } c0 = (q, \text{AR_SimRead}, 0, 0, buf0, dvec, posk0)$
and $tcorr: \forall k < k_tm \ M. \text{ar_tape_correspondence } (\Gamma_tm \ M) (\text{le_tm } M)$
 $(\text{bl_tm } M)$
 $(\text{mt_tape } cM \ k) (\text{mt_tape } c0 \ k)$
and $ppos: \bigwedge k. \text{mt_pos } c0 \ k = \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos}$
 $cM \ k)$
and $pkok: \bigwedge k. \text{posk0 } k = \text{AR_AtLE} \longleftrightarrow \text{mt_pos } cM \ k = 0$
and $tapeG: \bigwedge k. \text{mt_tape } cM \ k (\text{mt_pos } cM \ k) \in \Gamma_tm \ M$
and $bufG: \bigwedge k. \text{buf0 } k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\}$
and $pad0: \forall j \geq k_tm \ M. \text{mt_tape } c0 \ j (\text{mt_pos } c0 \ j) = \text{BLANK4}$
and $src0: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimRead}, 0, 0, buf0, dvec, posk0)$

shows $j \leq k_tm\ M - 1 \implies$
 $(\exists c\ m. (c0, c) \in R' \wedge m$
 $\wedge m \leq j * (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c = (q, AR_SimRead, k_unidx\ j, 0,$
 $(\lambda k. \text{if } k_idx\ k < j \text{ then } mt_tape\ cM\ k\ (mt_pos\ cM\ k) \text{ else } buf0\ k),$
 $dvec,$
 $(\lambda k. \text{if } k_idx\ k < j$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } AR_AtLE$
 $\text{ else if } mt_pos\ cM\ k = 1 \text{ then } AR_AtFirstProper$
 $\text{ else } AR_AtFurtherProper)$
 $\text{ else } posk0\ k))$
 $\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. \text{if } k_idx\ k < j$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } Suc\ 0$
 $\text{ else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
 $\text{ + } block_width\ (\Gamma_tm\ M))$
 $\text{ else } mt_pos\ c0\ k))$
 $\langle proof \rangle$

lemma *ar_read_prefix*:

fixes $M :: ('q, 'a)\ mttm$
and $cM :: ('a, 'q)\ mt_config$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0: mt_state\ c0 = (q, AR_SimRead, 0, 0, buf0, dvec, posk0)$
and $tcorr: \forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)$
 $(bl_tm\ M)$
 $(mt_tape\ cM\ k)\ (mt_tape\ c0\ k)$
and $ppos: \bigwedge k. mt_pos\ c0\ k = sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
and $pkok: \bigwedge k. posk0\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0$
and $tapeG: \bigwedge k. mt_tape\ cM\ k\ (mt_pos\ cM\ k) \in \Gamma_tm\ M$
and $bufG: \bigwedge k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, 0, 0, buf0, dvec, posk0)$
shows $j \leq k_tm\ M - 1 \implies$
 $(\exists c\ m. (c0, c) \in (mttm_step\ (alphabet_reduce_delta\ M)) \wedge m$
 $\wedge m \leq j * (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c = (q, AR_SimRead, k_unidx\ j, 0,$
 $(\lambda k. \text{if } k_idx\ k < j \text{ then } mt_tape\ cM\ k\ (mt_pos\ cM\ k) \text{ else } buf0\ k),$
 $dvec,$
 $(\lambda k. \text{if } k_idx\ k < j$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } AR_AtLE$
 $\text{ else if } mt_pos\ cM\ k = 1 \text{ then } AR_AtFirstProper$
 $\text{ else } AR_AtFurtherProper)$
 $\text{ else } posk0\ k))$

$\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. \text{if } k_idx\ k < j$
 $\quad \text{then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } Suc\ 0$
 $\quad \quad \text{else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
 $\quad \quad \quad + block_width\ (\Gamma_tm\ M))$
 $\quad \text{else } mt_pos\ c0\ k))$
 $\langle proof \rangle$

The full read phase: the prefix walk over the first $k_tm\ M - 1$ tapes followed by the unified last-tape read $ar_read_tape_finish_step$, landing at $AR_SimCompute$ (current-tape field $k_unidx\ 0$) with every tape's M -read symbol decoded into buf , every $posk$ re-synced exact, and every head left at its block-end (or position 1 for an LE tape). Aggregate cost $\leq k_tm\ M \cdot (b + 2)$. The final $buf/posk/$ position vectors collapse from the $k_idx\ k < j$ split because, on the last tape, every other tape has index $< k_tm\ M - 1$ (k_idx bijective into $\{..< k_tm\ M\}$).

Relation-generic full read phase scaffold. The two helpers (prefix walk, last-tape finish) are the $leaf_prefix$ and $leaf_finish$ hypotheses; union and sub versions are thin instantiations. The finish leaf is function-heavy, so it is discharged via ($rule\ \dots; assumption$) rather than positional OF .

lemma $ar_read_phase_gen$:

fixes $M :: ('q, 'a)\ mttm$
and $cM :: ('a, 'q)\ mt_config$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $R' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config\ rel$
assumes $leaf_prefix$:
 $\bigwedge jj. \llbracket valid_mttm\ M; q \in Q_tm\ M; 2 \leq block_width\ (\Gamma_tm\ M);$
 $\quad mt_state\ c0 = (q, AR_SimRead, 0, 0, buf0, dvec, posk0);$
 $\quad \forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)$
 $(bl_tm\ M)$
 $\quad (mt_tape\ cM\ k)\ (mt_tape\ c0\ k);$
 $\bigwedge k. mt_pos\ c0\ k = sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM$
 $k);$
 $\bigwedge k. posk0\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0;$
 $\bigwedge k. mt_tape\ cM\ k\ (mt_pos\ cM\ k) \in \Gamma_tm\ M;$
 $\bigwedge k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$
 $\forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, 0, 0, buf0, dvec, posk0);$
 $jj \leq k_tm\ M - 1 \rrbracket$
 $\implies (\exists c\ m. (c0, c) \in R' \wedge m$
 $\quad \wedge m \leq jj * (block_width\ (\Gamma_tm\ M) + 2)$
 $\quad \wedge mt_state\ c = (q, AR_SimRead, k_unidx\ jj, 0,$
 $\quad \quad (\lambda k. \text{if } k_idx\ k < jj \text{ then } mt_tape\ cM\ k\ (mt_pos\ cM\ k) \text{ else}$
 $buf0\ k),$
 $\quad dvec,$
 $\quad (\lambda k. \text{if } k_idx\ k < jj$
 $\quad \quad \text{then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } AR_AtLE$

$$\begin{aligned}
& \text{else if } mt_pos\ cM\ k = 1 \text{ then } AR_AtFirstProper \\
& \text{else } AR_AtFurtherProper) \\
& \text{else } posk0\ k)) \\
& \wedge mt_tape\ c = mt_tape\ c0 \\
& \wedge mt_pos\ c = (\lambda k. \text{if } k_idx\ k < jj \\
& \quad \text{then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } Suc\ 0 \\
& \quad \text{else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k) \\
& \quad \quad + block_width\ (\Gamma_tm\ M)) \\
& \quad \text{else } mt_pos\ c0\ k)) \\
& \text{and leaf_finish:} \\
& \quad \wedge cc\ tk'\ buf'\ posk'\ tM'\ p'. \\
& \quad \llbracket \text{valid_mttm } M; q \in Q_tm\ M; 2 \leq block_width\ (\Gamma_tm\ M); \\
& \quad \text{is_last_k } M\ tk'; \\
& \quad mt_state\ cc = (q, AR_SimRead, tk', 0, buf', dvec, posk'); \\
& \quad ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M) \\
& \quad \quad tM'\ (mt_tape\ cc\ tk'); \\
& \quad mt_pos\ cc\ tk' = sim_pos\ (block_width\ (\Gamma_tm\ M))\ p'; \\
& \quad posk'\ tk' = AR_AtLE \longleftrightarrow p' = 0; \\
& \quad 1 \leq p' \implies tM'\ p' \in \Gamma_tm\ M; \\
& \quad ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M) \\
& \quad \quad (AR_SimRead, tk', 0, buf', dvec, posk'); \\
& \quad \forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4; \\
& \quad ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M) \\
& \quad \quad (AR_SimRead, tk', 0, buf', dvec, posk') \rrbracket \\
& \implies \exists c''. (cc, c'') \in R' \wedge \wedge (\text{if } p' = 0 \text{ then } 1 \text{ else } block_width\ (\Gamma_tm\ M) \\
& + 2) \\
& \quad \wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0, \\
& \quad \quad buf'(tk' := tM'\ p'), dvec, \\
& \quad \quad posk'(tk' := \text{if } p' = 0 \text{ then } AR_AtLE \\
& \quad \quad \quad \text{else if } p' = 1 \text{ then } AR_AtFirstProper \\
& \quad \quad \quad \text{else } AR_AtFurtherProper)) \\
& \quad \wedge mt_tape\ c'' = mt_tape\ cc \\
& \quad \wedge mt_pos\ c'' = (mt_pos\ cc)(tk' := \\
& \quad \quad \text{if } p' = 0 \text{ then } Suc\ 0 \\
& \quad \quad \quad \text{else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ p' + block_width \\
& (\Gamma_tm\ M)) \\
& \quad \text{and } vM: \text{valid_mttm } M \\
& \quad \text{and } qQ: q \in Q_tm\ M \\
& \quad \text{and } kge2: 2 \leq block_width\ (\Gamma_tm\ M) \\
& \quad \text{and } stg0: mt_state\ c0 = (q, AR_SimRead, 0, 0, buf0, dvec, posk0) \\
& \quad \text{and } tcorr: \forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M) \\
& (bl_tm\ M) \\
& \quad \quad (mt_tape\ cM\ k)\ (mt_tape\ c0\ k) \\
& \quad \text{and } ppos: \wedge k. mt_pos\ c0\ k = sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos \\
& cM\ k) \\
& \quad \text{and } pkok: \wedge k. posk0\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0 \\
& \quad \text{and } tapeG: \wedge k. mt_tape\ cM\ k\ (mt_pos\ cM\ k) \in \Gamma_tm\ M \\
& \quad \text{and } bufG: \wedge k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\} \\
& \quad \text{and } pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4
\end{aligned}$$

and $src0$: $ar_stage_bounded$ (bl_tm M) (k_tm M)
 $(AR_SimRead, 0, 0, buf0, dvec, posk0)$
shows $\exists c\ m. (c0, c) \in R' \wedge \wedge m$
 $\wedge m \leq k_tm\ M * (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $(\lambda k. \text{if } k < k_tm\ M \text{ then } mt_tape\ cM\ k\ (mt_pos\ cM\ k) \text{ else } buf0\ k),$
 $dvec,$
 $(\lambda k. \text{if } k < k_tm\ M$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } AR_AtLE$
 $\text{ else if } mt_pos\ cM\ k = 1 \text{ then } AR_AtFirstProper$
 $\text{ else } AR_AtFurtherProper)$
 $\text{ else } posk0\ k))$
 $\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. \text{if } k < k_tm\ M$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } Suc\ 0$
 $\text{ else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k) + block_width$
 $(\Gamma_tm\ M))$
 $\text{ else } mt_pos\ c0\ k)$
 $\langle proof \rangle$

lemma ar_read_phase :

fixes $M :: ('q, 'a)\ mttm$
and $cM :: ('a, 'q)\ mt_config$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes vM : $valid_mttm\ M$
and qQ : $q \in Q_tm\ M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0$: $mt_state\ c0 = (q, AR_SimRead, 0, 0, buf0, dvec, posk0)$
and $tcrr$: $\forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)$
 $(bl_tm\ M)$
 $(mt_tape\ cM\ k)\ (mt_tape\ c0\ k)$
and $ppos$: $\bigwedge k. mt_pos\ c0\ k = sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
and $pkok$: $\bigwedge k. posk0\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0$
and $tapeG$: $\bigwedge k. mt_tape\ cM\ k\ (mt_pos\ cM\ k) \in \Gamma_tm\ M$
and $bufG$: $\bigwedge k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pad0$: $\forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0$: $ar_stage_bounded$ (bl_tm M) (k_tm M)
 $(AR_SimRead, 0, 0, buf0, dvec, posk0)$
shows $\exists c\ m. (c0, c) \in (mttm_step\ (alphabet_reduce_delta\ M)) \wedge \wedge m$
 $\wedge m \leq k_tm\ M * (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $(\lambda k. \text{if } k < k_tm\ M \text{ then } mt_tape\ cM\ k\ (mt_pos\ cM\ k) \text{ else } buf0\ k),$
 $dvec,$
 $(\lambda k. \text{if } k < k_tm\ M$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } AR_AtLE$
 $\text{ else if } mt_pos\ cM\ k = 1 \text{ then } AR_AtFirstProper$
 $\text{ else } AR_AtFurtherProper)$
 $\text{ else } posk0\ k))$

```

       $\wedge$   $mt\_tape\ c = mt\_tape\ c0$ 
       $\wedge$   $mt\_pos\ c = (\lambda k. \text{if } k < k\_tm\ M$ 
        then  $(\text{if } mt\_pos\ cM\ k = 0 \text{ then } Suc\ 0$ 
          else  $sim\_pos\ (block\_width\ (\Gamma\_tm\ M))\ (mt\_pos\ cM\ k) + block\_width$ 
             $(\Gamma\_tm\ M))$ 
          else  $mt\_pos\ c0\ k)$ 
       $\langle proof \rangle$ 

```

```

end
theory AlphabetReduction_ForwardWrite
  imports AlphabetReduction_ForwardRead
begin

```

2.14 Write phase composition

Relation-generic write back-walk loop scaffold. The walk leaf ($ar_write_walk_step$) is the *leaf* hypothesis; the union- and sub-relation back-loops are thin instantiations. Throughout, b abbreviates $block_width\ \Gamma$ (the per-symbol cell width).

lemma $ar_write_back_loop_gen$:

```

  fixes  $M :: ('q, 'a) mttm$ 
  and  $c0 :: (sym4, 'q \times 'a\ ar\_stage)\ mt\_config$ 
  and  $R' :: (sym4, 'q \times 'a\ ar\_stage)\ mt\_config\ rel$ 
  assumes leaf:
     $\bigwedge cc\ i. \llbracket valid\_mttm\ M;$ 
       $mt\_state\ cc = (q, AR\_SimWrite, tk, i, buf, dvec, posk);$ 
       $q \in Q\_tm\ M;$ 
       $posk\ tk \neq AR\_AtLE;$ 
       $mt\_tape\ cc\ tk\ (mt\_pos\ cc\ tk) \neq LE4;$ 
       $Suc\ i \leq block\_width\ (\Gamma\_tm\ M);$ 
       $ar\_valid\_stage\ (\Gamma\_tm\ M)\ (bl\_tm\ M)$ 
         $(AR\_SimWrite, tk, i, buf, dvec, posk);$ 
       $\forall j \geq k\_tm\ M. mt\_tape\ cc\ j\ (mt\_pos\ cc\ j) = BLANK4;$ 
       $ar\_stage\_bounded\ (bl\_tm\ M)\ (k\_tm\ M)$ 
         $(AR\_SimWrite, tk, i, buf, dvec, posk) \rrbracket$ 
     $\implies \exists c''. (cc, c'') \in R'$ 
       $\wedge mt\_state\ c'' = (q, AR\_SimWrite, tk, Suc\ i, buf, dvec,$ 
 $posk)$ 
       $\wedge mt\_tape\ c'' = mt\_tape\ cc$ 
       $\wedge mt\_pos\ c'' = (mt\_pos\ cc)(tk := mt\_pos\ cc\ tk - 1)$ 
  and  $vM: valid\_mttm\ M$ 
  and  $qQ: q \in Q\_tm\ M$ 
  and  $kge2: 2 \leq block\_width\ (\Gamma\_tm\ M)$ 
  and  $poskproper: posk\ tk \neq AR\_AtLE$ 
  and  $stg: mt\_state\ c0 = (q, AR\_SimWrite, tk, 0, buf, dvec, posk)$ 
  and  $buf\_valid: \forall k. buf\ k \in \Gamma\_tm\ M \cup \{bl\_tm\ M\}$ 
  and  $pos\_base: mt\_pos\ c0\ tk = base + block\_width\ (\Gamma\_tm\ M)$ 
  and  $notLE: \bigwedge m. \llbracket 1 \leq m; m \leq block\_width\ (\Gamma\_tm\ M) \rrbracket$ 
     $\implies mt\_tape\ c0\ tk\ (base + m) \neq LE4$ 

```

and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in R' \wedge \wedge block_width\ (\Gamma_tm\ M)$
 $\wedge mt_state\ c' = (q, AR_SimWrite, tk, block_width\ (\Gamma_tm\ M),$
 $buf, dvec, posk)$
 $\wedge mt_pos\ c'\ tk = base$
 $\wedge mt_tape\ c' = mt_tape\ c0$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c'\ k' = mt_pos\ c0\ k')$
 $\langle proof \rangle$

The write back-walk loop, proper tape. From an $AR_SimWrite$ stage at bit-counter 0 with the head at the block end $base + b$ (where the read phase left it) and $buf\ tk$ a proper symbol, the head walks L across the b -cell block back to $base = sim_pos\ p$, reaching bit-counter b . Tape unchanged (the back-walk only moves); this is the simpler of the two write loops, the analogue of $ar_read_bit_loop$ with a constant-tape invariant. Each $ar_write_walk_step$ needs its current cell $\neq LE4$; the visited cells are $base + 1 \dots base + b$ (all proper positions), supplied by $notLE$. Chain length b .

lemma $ar_write_back_loop:$
fixes $M :: ('q, 'a)\ mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $poskproper: posk\ tk \neq AR_AtLE$
and $stg: mt_state\ c0 = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pos_base: mt_pos\ c0\ tk = base + block_width\ (\Gamma_tm\ M)$
and $notLE: \bigwedge m. \llbracket 1 \leq m; m \leq block_width\ (\Gamma_tm\ M) \rrbracket$
 $\implies mt_tape\ c0\ tk\ (base + m) \neq LE4$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in (mttm_step\ (alphabet_reduce_delta\ M))$
 $\wedge \wedge block_width\ (\Gamma_tm\ M)$
 $\wedge mt_state\ c' = (q, AR_SimWrite, tk, block_width\ (\Gamma_tm\ M),$
 $buf, dvec, posk)$
 $\wedge mt_pos\ c'\ tk = base$
 $\wedge mt_tape\ c' = mt_tape\ c0$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c'\ k' = mt_pos\ c0\ k')$
 $\langle proof \rangle$

The write forward bit-write loop, proper tape. From an $AR_SimWrite$ stage at bit-counter b with the head back at $base$ (where $ar_write_back_loop$ left it), the head walks R across the block writing each cell, reaching bit-counter $b + (b - 1)$ with the head at $base + (b - 1)$. Unlike the read

loops this loop *mutates* the tape: its invariant carries a partially-overwritten tape (cells $base \dots base + j - 1$ already hold the new *write_bit* image, the rest hold old content), and each *ar_write_bit_step*'s $\neq LE4$ guard reads the *not-yet-written* current cell $base + j$, which still holds old content (supplied $\neq LE4$ by *notLE*). Writes the first $b - 1$ cells; the last is the boundary step's job. Chain length $b - 1$.

Relation-generic write forward-walk loop scaffold. The bit-writing leaf (*ar_write_bit_step*) is the *leaf* hypothesis; the union- and sub-relation forward-loops are thin instantiations.

lemma *ar_write_fwd_loop_gen*:

fixes $M :: ('q, 'a) mttm$
and $c0 :: (sym4, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
and $R' :: (sym4, 'q \times 'a \text{ ar_stage}) \text{ mt_config rel}$
assumes *leaf*:
 $\bigwedge cc \ i. \llbracket \text{valid_mttm } M;$
 $\text{mt_state } cc = (q, AR_SimWrite, tk, i, buf, dvec, posk);$
 $q \in Q_tm \ M;$
 $posk \ tk \neq AR_AtLE;$
 $\text{mt_tape } cc \ tk \ (\text{mt_pos } cc \ tk) \neq LE4;$
 $\text{block_width } (\Gamma_tm \ M) \leq i;$
 $Suc \ i < 2 * \text{block_width } (\Gamma_tm \ M);$
 $\text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk);$
 $\forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = BLANK4;$
 $\text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge \text{mt_state } c'' = (q, AR_SimWrite, tk, Suc \ i, buf, dvec,$
 $posk)$
 $\wedge \text{mt_tape } c'' = (\text{mt_tape } cc)(tk :=$
 $(\text{mt_tape } cc \ tk)(\text{mt_pos } cc \ tk$
 $:= \text{write_bit } (\Gamma_tm \ M) \ (\text{bl_tm } M) \ (buf \ tk)$
 $(i - \text{block_width } (\Gamma_tm \ M))))$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } cc)(tk := Suc \ (\text{mt_pos } cc \ tk))$
and $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $poskproper: posk \ tk \neq AR_AtLE$
and $stg: \text{mt_state } c0 = (q, AR_SimWrite, tk, \text{block_width } (\Gamma_tm \ M),$
 $buf, dvec, posk)$
and $buf_valid: \forall k. buf \ k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\}$
and $pos_base: \text{mt_pos } c0 \ tk = base$
and $notLE: \bigwedge m. m < \text{block_width } (\Gamma_tm \ M)$
 $\implies \text{mt_tape } c0 \ tk \ (base + m) \neq LE4$
and $pad0: \forall j \geq k_tm \ M. \text{mt_tape } c0 \ j \ (\text{mt_pos } c0 \ j) = BLANK4$
and $src0: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(AR_SimWrite, tk, \text{block_width } (\Gamma_tm \ M), buf, dvec, posk)$
shows $\exists c'. (c0, c') \in R' \wedge \wedge (\text{block_width } (\Gamma_tm \ M) - 1)$

$\wedge \text{mt_state } c' = (q, \text{AR_SimWrite}, tk,$
 $\quad \text{block_width } (\Gamma_tm\ M) + (\text{block_width } (\Gamma_tm\ M) - 1),$
 $\quad \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c' tk = \text{base} + (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\wedge \text{mt_tape } c' tk = (\lambda pos.$
 $\quad \text{if } \text{base} \leq pos \wedge pos < \text{base} + (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\quad \text{then write_bit } (\Gamma_tm\ M) (\text{bl_tm } M) (\text{buf } tk) (pos - \text{base})$
 $\quad \text{else mt_tape } c0\ tk\ pos)$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_tape } c' k' = \text{mt_tape } c0\ k')$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c' k' = \text{mt_pos } c0\ k')$
 $\langle \text{proof} \rangle$

lemma *ar_write_fwd_loop*:

fixes $M :: ('q, 'a) \text{mttm}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm\ M)$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{stg}: \text{mt_state } c0 = (q, \text{AR_SimWrite}, tk, \text{block_width } (\Gamma_tm\ M),$
 $\quad \text{buf}, \text{dvec}, \text{posk})$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm\ M \cup \{\text{bl_tm } M\}$
and $\text{pos_base}: \text{mt_pos } c0\ tk = \text{base}$
and $\text{notLE}: \bigwedge m. m < \text{block_width } (\Gamma_tm\ M)$
 $\implies \text{mt_tape } c0\ tk\ (\text{base} + m) \neq \text{LE4}$
and $\text{pad0}: \forall j \geq k_tm\ M. \text{mt_tape } c0\ j\ (\text{mt_pos } c0\ j) = \text{BLANK4}$
and $\text{src0}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm\ M)$
 $(\text{AR_SimWrite}, tk, \text{block_width } (\Gamma_tm\ M), \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c'. (c0, c') \in (\text{mttm_step } (\text{alphabet_reduce_delta } M))$
 $\quad \wedge \wedge (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\wedge \text{mt_state } c' = (q, \text{AR_SimWrite}, tk,$
 $\quad \text{block_width } (\Gamma_tm\ M) + (\text{block_width } (\Gamma_tm\ M) - 1),$
 $\quad \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c' tk = \text{base} + (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\wedge \text{mt_tape } c' tk = (\lambda pos.$
 $\quad \text{if } \text{base} \leq pos \wedge pos < \text{base} + (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\quad \text{then write_bit } (\Gamma_tm\ M) (\text{bl_tm } M) (\text{buf } tk) (pos - \text{base})$
 $\quad \text{else mt_tape } c0\ tk\ pos)$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_tape } c' k' = \text{mt_tape } c0\ k')$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c' k' = \text{mt_pos } c0\ k')$
 $\langle \text{proof} \rangle$

The proper-cell single-tape write prefix: the part of a proper-cell single-tape write shared by the non-last (*boundary*) and last (*finish*) variants. From an *AR_SimWrite* stage at bit-counter 0 with the head at the block end $\text{base} + b$ and $\text{buf } tk$ a proper symbol, it fires the back-walk loop (*ar_write_back_loop*, head to base) and the forward bit-write loop (*ar_write_fwd_loop*, writing the first $b - 1$ cells), reaching bit-counter $b + (b - 1)$ with the head at $\text{base} + (b - 1)$. The remaining last-cell step

(boundary or finish) is added by the consumer. The two segment costs compose by *relpow_add* on *relcompI* (chain length $b + (b - 1)$). The *notLE* hypothesis (every block cell and the next-block lead cell $\neq LE4$) feeds both loops.

Relation-generic write proper-prefix scaffold. The two loop helpers (back, forward) are the *leaf_back* and *leaf_fwd* hypotheses; the union- and sub-relation prefixes are thin instantiations.

lemma *ar_write_proper_prefix_gen*:

fixes $M :: ('q, 'a) \text{ mttm}$

and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$

and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config rel}$

assumes *leaf_back*:

$\wedge cc. \llbracket \text{valid_mttm } M; q \in Q_tm \ M; 2 \leq \text{block_width } (\Gamma_tm \ M);$
 $\text{posk } tk \neq AR_AtLE;$
 $\text{mt_state } cc = (q, AR_SimWrite, tk, 0, \text{buf}, \text{dvec}, \text{posk});$
 $\forall k. \text{buf } k \in \Gamma_tm \ M \cup \{bl_tm \ M\};$
 $\text{mt_pos } cc \ tk = \text{base} + \text{block_width } (\Gamma_tm \ M);$
 $\wedge m. \llbracket 1 \leq m; m \leq \text{block_width } (\Gamma_tm \ M) \rrbracket$
 $\implies \text{mt_tape } cc \ tk \ (\text{base} + m) \neq LE4;$
 $\forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = BLANK4;$
 $\text{ar_stage_bounded } (bl_tm \ M) \ (k_tm \ M)$
 $(AR_SimWrite, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \rrbracket$
 $\implies \exists c1. (cc, c1) \in R' \wedge \wedge \text{block_width } (\Gamma_tm \ M)$
 $\wedge \text{mt_state } c1 = (q, AR_SimWrite, tk, \text{block_width } (\Gamma_tm \ M),$
 $\text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c1 \ tk = \text{base}$
 $\wedge \text{mt_tape } c1 = \text{mt_tape } cc$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c1 \ k' = \text{mt_pos } cc \ k')$

and *leaf_fwd*:

$\wedge cc. \llbracket \text{valid_mttm } M; q \in Q_tm \ M; 2 \leq \text{block_width } (\Gamma_tm \ M);$
 $\text{posk } tk \neq AR_AtLE;$
 $\text{mt_state } cc = (q, AR_SimWrite, tk, \text{block_width } (\Gamma_tm \ M), \text{buf},$
 $\text{dvec}, \text{posk});$
 $\forall k. \text{buf } k \in \Gamma_tm \ M \cup \{bl_tm \ M\};$
 $\text{mt_pos } cc \ tk = \text{base};$
 $\wedge m. m < \text{block_width } (\Gamma_tm \ M) \implies \text{mt_tape } cc \ tk \ (\text{base} + m) \neq$
 $LE4;$
 $\forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = BLANK4;$
 $\text{ar_stage_bounded } (bl_tm \ M) \ (k_tm \ M)$
 $(AR_SimWrite, tk, \text{block_width } (\Gamma_tm \ M), \text{buf}, \text{dvec}, \text{posk}) \rrbracket$
 $\implies \exists c2. (cc, c2) \in R' \wedge \wedge (\text{block_width } (\Gamma_tm \ M) - 1)$
 $\wedge \text{mt_state } c2 = (q, AR_SimWrite, tk,$
 $\text{block_width } (\Gamma_tm \ M) + (\text{block_width } (\Gamma_tm \ M) - 1),$
 $\text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c2 \ tk = \text{base} + (\text{block_width } (\Gamma_tm \ M) - 1)$
 $\wedge \text{mt_tape } c2 \ tk = (\lambda pos.$
 $\text{if } \text{base} \leq pos \wedge pos < \text{base} + (\text{block_width } (\Gamma_tm \ M) - 1)$
 $\text{then } \text{write_bit } (\Gamma_tm \ M) \ (bl_tm \ M) \ (\text{buf } tk) \ (pos - \text{base})$

$\text{else } mt_tape\ cc\ tk\ pos)$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_tape\ c2\ k' = mt_tape\ cc\ k')$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c2\ k' = mt_pos\ cc\ k')$
and vM : $valid_mttm\ M$
and qQ : $q \in Q_tm\ M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm\ M)$
and $poskproper$: $posk\ tk \neq AR_AtLE$
and stg : $mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and buf_valid : $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and pos_base : $mt_pos\ c'\ tk = base + block_width\ (\Gamma_tm\ M)$
and $notLE$: $\bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$
 $\implies mt_tape\ c'\ tk\ (base + m) \neq LE4$
and $pad0$: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0$: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c2. (c', c2) \in R' \wedge (block_width\ (\Gamma_tm\ M) + (block_width\ (\Gamma_tm\ M) - 1))$
 $\wedge mt_state\ c2 = (q, AR_SimWrite, tk,$
 $block_width\ (\Gamma_tm\ M) + (block_width\ (\Gamma_tm\ M) - 1),$
 $buf, dvec, posk)$
 $\wedge mt_tape\ c2 = (mt_tape\ c')(tk := (\lambda pos.$
 $if\ base \leq pos \wedge pos < base + (block_width\ (\Gamma_tm\ M) - 1)$
 $then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos - base)$
 $else\ mt_tape\ c'\ tk\ pos))$
 $\wedge mt_pos\ c2 = (mt_pos\ c')(tk := base + (block_width\ (\Gamma_tm\ M) -$
 $1))$
 $\langle proof \rangle$

lemma $ar_write_proper_prefix$:

fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes vM : $valid_mttm\ M$
and qQ : $q \in Q_tm\ M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm\ M)$
and $poskproper$: $posk\ tk \neq AR_AtLE$
and stg : $mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and buf_valid : $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and pos_base : $mt_pos\ c'\ tk = base + block_width\ (\Gamma_tm\ M)$
and $notLE$: $\bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$
 $\implies mt_tape\ c'\ tk\ (base + m) \neq LE4$
and $pad0$: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0$: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c2. (c', c2) \in (mttm_step\ (alphabet_reduce_delta\ M))$
 $\wedge (block_width\ (\Gamma_tm\ M) + (block_width\ (\Gamma_tm\ M) -$
 $1))$
 $\wedge mt_state\ c2 = (q, AR_SimWrite, tk,$
 $block_width\ (\Gamma_tm\ M) + (block_width\ (\Gamma_tm\ M) - 1),$
 $buf, dvec, posk)$

$\wedge \text{mt_tape } c2 = (\text{mt_tape } c')(tk := (\lambda pos.$
 $\quad \text{if } base \leq pos \wedge pos < base + (\text{block_width } (\Gamma_tm M) - 1)$
 $\quad \text{then write_bit } (\Gamma_tm M) (bl_tm M) (buf tk) (pos - base)$
 $\quad \text{else mt_tape } c' tk pos))$
 $\wedge \text{mt_pos } c2 = (\text{mt_pos } c')(tk := base + (\text{block_width } (\Gamma_tm M) -$
 $1))$
 $\langle proof \rangle$

Extend a written block prefix by its last cell. Both proper-write finishers reach a tape whose first $b - 1$ block cells already hold the new *write_bit* image; the boundary / finish step writes the last cell $base + (b - 1)$. This *fun_upd* closes the gap: updating the $b - 1$ -prefix function at $base + (b - 1)$ with $f (b - 1)$ yields the full b -cell write. Pure list/function arithmetic, shared by *ar_write_proper_step* and *ar_write_proper_finish_step*.

lemma *write_block_extend*:

fixes $k \text{ base} :: \text{nat}$ **and** $f g :: \text{nat} \Rightarrow \text{sym4}$

assumes $0 < k$

shows $((\lambda pos. \text{if } base \leq pos \wedge pos < base + (k - 1)$
 $\quad \text{then } f (pos - base) \text{ else } g pos)$
 $\quad (base + (k - 1) := f (k - 1)))$
 $= (\lambda pos. \text{if } base \leq pos \wedge pos < base + k$
 $\quad \text{then } f (pos - base) \text{ else } g pos)$

$\langle proof \rangle$

The proper-cell single-tape write, non-last tape: the prefix followed by the bit-boundary step (*ar_write_bit_boundary_step*), writing the last block cell $base + (b - 1)$ and landing at the next tape's *AR_SimWrite* ($k_succ tk$, counter 0). After the $2b$ -step walk the b -cell block at $base$ spells out *write_bit* ($buf tk$) (the encoded new symbol), every other cell unchanged, and the head returns to the block end $base + b$. Chain length $2b$.

Relation-generic write proper-step scaffold. The prefix walk and the closing bit-boundary step are the *leaf_prefix* and *leaf_boundary* hypotheses; union and sub versions are thin instantiations.

lemma *ar_write_proper_step_gen*:

fixes $M :: ('q, 'a) \text{mttm}$

and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$

and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config rel}$

assumes *leaf_prefix*:

$\wedge cc. \llbracket \text{valid_mttm } M; q \in Q_tm M; 2 \leq \text{block_width } (\Gamma_tm M);$
 $\text{posk } tk \neq \text{AR_AtLE};$
 $\text{mt_state } cc = (q, \text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk});$
 $\forall k. \text{buf } k \in \Gamma_tm M \cup \{bl_tm M\};$
 $\text{mt_pos } cc tk = base + \text{block_width } (\Gamma_tm M);$
 $\wedge m. m \leq \text{block_width } (\Gamma_tm M)$
 $\implies \text{mt_tape } cc tk (base + m) \neq \text{LE4};$
 $\forall j \geq k_tm M. \text{mt_tape } cc j (\text{mt_pos } cc j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (bl_tm M) (k_tm M)$

$(AR_SimWrite, tk, 0, buf, dvec, posk) \parallel$
 $\implies \exists c2. (cc, c2) \in R' \wedge (block_width (\Gamma_tm M) + (block_width$
 $(\Gamma_tm M) - 1))$
 $\wedge mt_state c2 = (q, AR_SimWrite, tk,$
 $block_width (\Gamma_tm M) + (block_width (\Gamma_tm M) - 1),$
 $buf, dvec, posk)$
 $\wedge mt_tape c2 = (mt_tape cc)(tk := (\lambda pos.$
 $if base \leq pos \wedge pos < base + (block_width (\Gamma_tm M) - 1)$
 $then write_bit (\Gamma_tm M) (bl_tm M) (buf tk) (pos - base)$
 $else mt_tape cc tk pos))$
 $\wedge mt_pos c2 = (mt_pos cc)(tk := base + (block_width (\Gamma_tm$
 $M) - 1))$
and leaf_boundary:
 $\wedge cc i. \parallel valid_mttm M;$
 $mt_state cc = (q, AR_SimWrite, tk, i, buf, dvec, posk);$
 $q \in Q_tm M;$
 $posk tk \neq AR_AtLE;$
 $mt_tape cc tk (mt_pos cc tk) \neq LE4;$
 $\neg is_last_k M tk;$
 $Suc i = 2 * block_width (\Gamma_tm M);$
 $ar_valid_stage (\Gamma_tm M) (bl_tm M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk);$
 $\forall j \geq k_tm M. mt_tape cc j (mt_pos cc j) = BLANK4;$
 $ar_stage_bounded (bl_tm M) (k_tm M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk) \parallel$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge mt_state c'' = (q, AR_SimWrite, k_succ tk, 0, buf, dvec,$
 $posk)$
 $\wedge mt_tape c'' = (mt_tape cc)(tk :=$
 $(mt_tape cc tk)(mt_pos cc tk :=$
 $write_bit (\Gamma_tm M) (bl_tm M) (buf tk)$
 $(i - block_width (\Gamma_tm M))))$
 $\wedge mt_pos c'' = (mt_pos cc)(tk := Suc (mt_pos cc tk))$
and vM: valid_mttm M
and qQ: $q \in Q_tm M$
and kge2: $2 \leq block_width (\Gamma_tm M)$
and notlast: $\neg is_last_k M tk$
and poskproper: $posk tk \neq AR_AtLE$
and stg: $mt_state c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and buf_valid: $\forall k. buf k \in \Gamma_tm M \cup \{bl_tm M\}$
and pos_base: $mt_pos c' tk = base + block_width (\Gamma_tm M)$
and notLE: $\wedge m. m \leq block_width (\Gamma_tm M)$
 $\implies mt_tape c' tk (base + m) \neq LE4$
and pad0: $\forall j \geq k_tm M. mt_tape c' j (mt_pos c' j) = BLANK4$
and src0: $ar_stage_bounded (bl_tm M) (k_tm M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge (2 * block_width (\Gamma_tm M))$
 $\wedge mt_state c'' = (q, AR_SimWrite, k_succ tk, 0, buf, dvec, posk)$
 $\wedge mt_tape c'' = (mt_tape c')(tk := (\lambda pos.$

$$\begin{aligned} & \text{if } \text{base} \leq \text{pos} \wedge \text{pos} < \text{base} + \text{block_width } (\Gamma_tm\ M) \\ & \text{then } \text{write_bit } (\Gamma_tm\ M) (\text{bl_tm } M) (\text{buf } tk) (\text{pos} - \text{base}) \\ & \text{else } \text{mt_tape } c' tk\ \text{pos}) \\ & \wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \text{base} + \text{block_width } (\Gamma_tm\ M)) \end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *ar_write_proper_step*:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (\text{sym4}, 'q \times 'a\ \text{ar_stage})\ \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm\ M)$
and $\text{notlast}: \neg \text{is_last_k } M\ tk$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm\ M \cup \{\text{bl_tm } M\}$
and $\text{pos_base}: \text{mt_pos } c' tk = \text{base} + \text{block_width } (\Gamma_tm\ M)$
and $\text{notLE}: \bigwedge m. m \leq \text{block_width } (\Gamma_tm\ M)$
 $\implies \text{mt_tape } c' tk (\text{base} + m) \neq \text{LE4}$
and $\text{pad0}: \forall j \geq k_tm\ M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src0}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm\ M)$
 $(\text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in (\text{mttm_step } (\text{alphabet_reduce_delta } M))$
 $\wedge (2 * \text{block_width } (\Gamma_tm\ M))$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimWrite}, k_succ\ tk, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = (\text{mt_tape } c')(tk := (\lambda \text{pos.}$
 $\text{if } \text{base} \leq \text{pos} \wedge \text{pos} < \text{base} + \text{block_width } (\Gamma_tm\ M)$
 $\text{then } \text{write_bit } (\Gamma_tm\ M) (\text{bl_tm } M) (\text{buf } tk) (\text{pos} - \text{base})$
 $\text{else } \text{mt_tape } c' tk\ \text{pos}))$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \text{base} + \text{block_width } (\Gamma_tm\ M))$
 $\langle \text{proof} \rangle$

The proper-cell single-tape write, last tape: as *ar_write_proper_step* but tk is the last tape, so after the last-cell write (*ar_write_bit_finish_step*) the phase transitions to *AR_SimAdvance* with the current-tape field reset to $k_unidx\ 0$ (all tapes written). Same b -cell block write and head return to $\text{base} + b$. Chain length $2b$.

Relation-generic write proper-finish-step scaffold. As *ar_write_proper_step_gen* but the closing leaf is the last-tape finish (transition to *AR_SimAdvance*); the prefix and finish helpers are the *leaf_prefix* and *leaf_finish* hypotheses.

lemma *ar_write_proper_finish_step_gen*:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (\text{sym4}, 'q \times 'a\ \text{ar_stage})\ \text{mt_config}$
and $R' :: (\text{sym4}, 'q \times 'a\ \text{ar_stage})\ \text{mt_config } rel$
assumes *leaf_prefix*:
 $\bigwedge cc. \llbracket \text{valid_mttm } M; q \in Q_tm\ M; 2 \leq \text{block_width } (\Gamma_tm\ M);$
 $\text{posk } tk \neq \text{AR_AtLE};$

$mt_state\ cc = (q, AR_SimWrite, tk, 0, buf, dvec, posk);$
 $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$
 $mt_pos\ cc\ tk = base + block_width\ (\Gamma_tm\ M);$
 $\bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$
 $\implies mt_tape\ cc\ tk\ (base + m) \neq LE4;$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk) \parallel$
 $\implies \exists c2. (cc, c2) \in R' \wedge (block_width\ (\Gamma_tm\ M) + (block_width$
 $(\Gamma_tm\ M) - 1))$
 $\wedge mt_state\ c2 = (q, AR_SimWrite, tk,$
 $block_width\ (\Gamma_tm\ M) + (block_width\ (\Gamma_tm\ M) - 1),$
 $buf, dvec, posk)$
 $\wedge mt_tape\ c2 = (mt_tape\ cc)(tk := (\lambda pos.$
 $if\ base \leq pos \wedge pos < base + (block_width\ (\Gamma_tm\ M) - 1)$
 $then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos - base)$
 $else\ mt_tape\ cc\ tk\ pos))$
 $\wedge mt_pos\ c2 = (mt_pos\ cc)(tk := base + (block_width\ (\Gamma_tm$
 $M) - 1))$
and $leaf_finish:$
 $\wedge cc\ i. \parallel valid_mttm\ M;$
 $mt_state\ cc = (q, AR_SimWrite, tk, i, buf, dvec, posk);$
 $q \in Q_tm\ M;$
 $posk\ tk \neq AR_AtLE;$
 $mt_tape\ cc\ tk\ (mt_pos\ cc\ tk) \neq LE4;$
 $is_last_k\ M\ tk;$
 $Suc\ i = 2 * block_width\ (\Gamma_tm\ M);$
 $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk);$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk) \parallel$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf,$
 $dvec, posk)$
 $\wedge mt_tape\ c'' = (mt_tape\ cc)(tk :=$
 $(mt_tape\ cc\ tk)(mt_pos\ cc\ tk :=$
 $write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $(i - block_width\ (\Gamma_tm\ M))))$
 $\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := Suc\ (mt_pos\ cc\ tk))$
and $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $last: is_last_k\ M\ tk$
and $poskproper: posk\ tk \neq AR_AtLE$
and $stg: mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pos_base: mt_pos\ c'\ tk = base + block_width\ (\Gamma_tm\ M)$
and $notLE: \bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$

$\implies mt_tape\ c'\ tk\ (base + m) \neq LE4$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = (mt_tape\ c')(tk := (\lambda pos.$
 $\quad if\ base \leq pos \wedge pos < base + block_width\ (\Gamma_tm\ M)$
 $\quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos - base)$
 $\quad else\ mt_tape\ c'\ tk\ pos))$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + block_width\ (\Gamma_tm\ M))$
 $\langle proof \rangle$

lemma $ar_write_proper_finish_step:$

fixes $M :: ('q, 'a)\ mttm$

and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$

assumes $vM: valid_mttm\ M$

and $qQ: q \in Q_tm\ M$

and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$

and $last: is_last_k\ M\ tk$

and $poskproper: posk\ tk \neq AR_AtLE$

and $stg: mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$

and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$

and $pos_base: mt_pos\ c'\ tk = base + block_width\ (\Gamma_tm\ M)$

and $notLE: \bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$

$\implies mt_tape\ c'\ tk\ (base + m) \neq LE4$

and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$

and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimWrite, tk, 0, buf, dvec, posk)$

shows $\exists c''. (c', c'') \in (mttm_step\ (alphabet_reduce_delta\ M))$

$\wedge (2 * block_width\ (\Gamma_tm\ M))$

$\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$

$\wedge mt_tape\ c'' = (mt_tape\ c')(tk := (\lambda pos.$

$\quad if\ base \leq pos \wedge pos < base + block_width\ (\Gamma_tm\ M)$

$\quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos - base)$

$\quad else\ mt_tape\ c'\ tk\ pos))$

$\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + block_width\ (\Gamma_tm\ M))$

$\langle proof \rangle$

The unified single-tape write, non-last tape: the LE/proper dispatch, the AR analogue of $ar_read_tape_step$. From an $AR_SimWrite$ stage at bit-counter 0, dispatch on $posk\ tk = AR_AtLE$ (which the caller pins to $p = 0$, M 's head on the marker): the LE arm ($ar_write_le_step$, 1 step) leaves the tape and head untouched and hands off; the proper arm ($ar_write_proper_step$, $2b$ steps) overwrites the b -cell block at $sim_pos\ p$ with $write_bit\ (buf\ tk)$. In both arms every head is unchanged (the LE arm is all- N ; the proper arm back-walks then returns to the block end), so the conclusion is $mt_pos\ c'' = mt_pos\ c'$ uniformly. The proper-arm $\neq LE4$ facts come from the input correspondence $tcrr$ (block- p cells and the

block- $(p+1)$ lead cell, all $cell_repr$ cells via $cell_repr_nth_not_LE4$).

Relation-generic write single-tape step scaffold (non-last). The two cases ($p = 0$ le-step, $p \neq 0$ proper-step) are the $leaf_le$ and $leaf_proper$ hypotheses; union and sub versions are thin instantiations.

lemma $ar_write_tape_step_gen$:

fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a ar_stage) mt_config$
and $tM :: nat \Rightarrow 'a$
and $R' :: (sym4, 'q \times 'a ar_stage) mt_config rel$
assumes $leaf_le$:
 $\wedge cc. \llbracket valid_mttm\ M;$
 $mt_state\ cc = (q, AR_SimWrite, tk, 0, buf, dvec, posk);$
 $q \in Q_tm\ M;$
 $posk\ tk = AR_AtLE;$
 $\neg is_last_k\ M\ tk;$
 $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk);$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge mt_state\ c'' = (q, AR_SimWrite, k_succ\ tk, 0, buf, dvec,$
 $posk)$
 $\wedge mt_tape\ c'' = mt_tape\ cc$
 $\wedge mt_pos\ c'' = mt_pos\ cc$
and $leaf_proper$:
 $\wedge cc\ base'. \llbracket valid_mttm\ M; q \in Q_tm\ M; 2 \leq block_width\ (\Gamma_tm\ M);$
 $\neg is_last_k\ M\ tk;$
 $posk\ tk \neq AR_AtLE;$
 $mt_state\ cc = (q, AR_SimWrite, tk, 0, buf, dvec, posk);$
 $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$
 $mt_pos\ cc\ tk = base' + block_width\ (\Gamma_tm\ M);$
 $\wedge m. m \leq block_width\ (\Gamma_tm\ M)$
 $\implies mt_tape\ cc\ tk\ (base' + m) \neq LE4;$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R' \wedge (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimWrite, k_succ\ tk, 0, buf,$
 $dvec, posk)$
 $\wedge mt_tape\ c'' = (mt_tape\ cc)(tk := (\lambda pos.$
 $if\ base' \leq pos \wedge pos < base' + block_width\ (\Gamma_tm\ M)$
 $then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos -$
 $base')$
 $else\ mt_tape\ cc\ tk\ pos))$
 $\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := base' + block_width$
 $(\Gamma_tm\ M))$
and $vM: valid_mttm\ M$

and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $notlast: \neg is_last_k\ M\ tk$
and $stg: mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and $tcrr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $\quad tM\ (mt_tape\ c'\ tk)$
and $ppos: mt_pos\ c'\ tk = (if\ p = 0\ then\ Suc\ 0$
 $\quad else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
and $poskle: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $vsr: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad (AR_SimWrite, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $\quad (AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in R' \wedge (if\ p = 0\ then\ 1\ else\ 2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimWrite, k_succ\ tk, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = (if\ p = 0\ then\ mt_tape\ c'$
 $\quad else\ (mt_tape\ c')(tk := (\lambda pos.$
 $\quad if\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p \leq pos$
 $\quad \wedge pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width$
 $(\Gamma_tm\ M)$
 $\quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $\quad (pos - sim_pos\ (block_width\ (\Gamma_tm\ M))\ p)$
 $\quad else\ mt_tape\ c'\ tk\ pos)))$
 $\wedge mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

lemma $ar_write_tape_step$:

fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $tM :: nat \Rightarrow 'a$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $notlast: \neg is_last_k\ M\ tk$
and $stg: mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and $tcrr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $\quad tM\ (mt_tape\ c'\ tk)$
and $ppos: mt_pos\ c'\ tk = (if\ p = 0\ then\ Suc\ 0$
 $\quad else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
and $poskle: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $vsr: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad (AR_SimWrite, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step (alphabet_reduce_delta M))$
 $\wedge (if\ p = 0\ then\ 1\ else\ 2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimWrite, k_succ\ tk, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = (if\ p = 0\ then\ mt_tape\ c'$
 $\quad else\ (mt_tape\ c')(tk := (\lambda pos.$
 $\quad if\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p \leq pos$
 $\quad \wedge pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width$
 $(\Gamma_tm\ M)$
 $\quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $\quad (pos - sim_pos\ (block_width\ (\Gamma_tm\ M))\ p)$
 $\quad else\ mt_tape\ c'\ tk\ pos)))$
 $\wedge mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

The unified single-tape write, last tape: as *ar_write_tape_step* but *tk* is the last tape, so both arms transition to *AR_SimAdvance* (current-tape field reset to *k_unidx 0*, all tapes written): the LE arm via *ar_write_le_finish_step*, the proper arm via *ar_write_proper_finish_step*. Same tape edit and head invariance.

Relation-generic write single-tape finish-step scaffold (last tape). As *ar_write_tape_step_gen* but both cases use the last-tape finish leaves (transition to *AR_SimAdvance*); the le-finish and proper-finish helpers are the *leaf_le* and *leaf_proper* hypotheses.

lemma *ar_write_tape_finish_step_gen*:

fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $tM :: nat \Rightarrow 'a$
and $R' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config\ rel$
assumes *leaf_le*:
 $\wedge cc. \llbracket valid_mttm\ M;$
 $mt_state\ cc = (q, AR_SimWrite, tk, 0, buf, dvec, posk);$
 $q \in Q_tm\ M;$
 $posk\ tk = AR_AtLE;$
 $is_last_k\ M\ tk;$
 $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk);$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec,$
 $posk)$
 $\wedge mt_tape\ c'' = mt_tape\ cc$
 $\wedge mt_pos\ c'' = mt_pos\ cc$
and *leaf_proper*:
 $\wedge cc\ base'. \llbracket valid_mttm\ M; q \in Q_tm\ M; 2 \leq block_width\ (\Gamma_tm\ M);$
 $is_last_k\ M\ tk;$

$posk\ tk \neq AR_AtLE;$
 $mt_state\ cc = (q, AR_SimWrite, tk, 0, buf, dvec, posk);$
 $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$
 $mt_pos\ cc\ tk = base' + block_width\ (\Gamma_tm\ M);$
 $\bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$
 $\implies mt_tape\ cc\ tk\ (base' + m) \neq LE4;$
 $\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk) \parallel$
 $\implies \exists c''. (cc, c'') \in R' \wedge (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf,$
 $dvec, posk)$
 $\wedge mt_tape\ c'' = (mt_tape\ cc)(tk := (\lambda pos.$
 $if\ base' \leq pos \wedge pos < base' + block_width\ (\Gamma_tm\ M)$
 $then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos -$
 $base')$
 $else\ mt_tape\ cc\ tk\ pos))$
 $\wedge mt_pos\ c'' = (mt_pos\ cc)(tk := base' + block_width$
 $(\Gamma_tm\ M))$
 $and\ vM: valid_mttm\ M$
 $and\ qQ: q \in Q_tm\ M$
 $and\ kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
 $and\ last: is_last_k\ M\ tk$
 $and\ stg: mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
 $and\ tcorr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $tM\ (mt_tape\ c'\ tk)$
 $and\ ppos: mt_pos\ c'\ tk = (if\ p = 0\ then\ Suc\ 0$
 $else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
 $and\ poskle: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
 $and\ buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
 $and\ usrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
 $and\ pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
 $and\ src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
 $shows\ \exists c''. (c', c'') \in R' \wedge (if\ p = 0\ then\ 1\ else\ 2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = (if\ p = 0\ then\ mt_tape\ c'$
 $else\ (mt_tape\ c')(tk := (\lambda pos.$
 $if\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p \leq pos$
 $\wedge pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width$
 $(\Gamma_tm\ M)$
 $then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $(pos - sim_pos\ (block_width\ (\Gamma_tm\ M))\ p)$
 $else\ mt_tape\ c'\ tk\ pos)))$
 $\wedge mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

```

lemma ar_write_tape_finish_step:
  fixes  $M :: ('q, 'a) \text{mttm}$ 
    and  $c' :: (\text{sym}_4, 'q \times 'a \text{ ar\_stage}) \text{mt\_config}$ 
    and  $tM :: \text{nat} \Rightarrow 'a$ 
  assumes  $vM: \text{valid\_mttm } M$ 
    and  $qQ: q \in Q\_tm \ M$ 
    and  $kge2: 2 \leq \text{block\_width } (\Gamma\_tm \ M)$ 
    and  $\text{last}: \text{is\_last\_k } M \ tk$ 
    and  $\text{stg}: \text{mt\_state } c' = (q, \text{AR\_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$ 
    and  $\text{tcrr}: \text{ar\_tape\_correspondence } (\Gamma\_tm \ M) (\text{le\_tm } M) (\text{bl\_tm } M)$ 
       $tM (\text{mt\_tape } c' \ tk)$ 
    and  $\text{ppos}: \text{mt\_pos } c' \ tk = (\text{if } p = 0 \text{ then } \text{Suc } 0$ 
       $\text{else } \text{sim\_pos } (\text{block\_width } (\Gamma\_tm \ M)) \ p + \text{block\_width } (\Gamma\_tm$ 
 $M))$ 
    and  $\text{poskle}: \text{posk } tk = \text{AR\_AtLE} \longleftrightarrow p = 0$ 
    and  $\text{buf\_valid}: \forall k. \text{buf } k \in \Gamma\_tm \ M \cup \{\text{bl\_tm } M\}$ 
    and  $\text{vsr}: \text{ar\_valid\_stage } (\Gamma\_tm \ M) (\text{bl\_tm } M)$ 
       $(\text{AR\_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$ 
    and  $\text{pad0}: \forall j \geq k\_tm \ M. \text{mt\_tape } c' \ j (\text{mt\_pos } c' \ j) = \text{BLANK}_4$ 
    and  $\text{src0}: \text{ar\_stage\_bounded } (\text{bl\_tm } M) (k\_tm \ M)$ 
       $(\text{AR\_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$ 
  shows  $\exists c''. (c', c'') \in (\text{mttm\_step } (\text{alphabet\_reduce\_delta } M))$ 
 $\wedge \wedge (\text{if } p = 0 \text{ then } 1 \text{ else } 2 * \text{block\_width } (\Gamma\_tm \ M))$ 
 $\wedge \text{mt\_state } c'' = (q, \text{AR\_SimAdvance}, k\_unidx \ 0, 0, \text{buf}, \text{dvec}, \text{posk})$ 
 $\wedge \text{mt\_tape } c'' = (\text{if } p = 0 \text{ then } \text{mt\_tape } c'$ 
 $\text{else } (\text{mt\_tape } c') (tk := (\lambda \text{pos.}$ 
 $\text{if } \text{sim\_pos } (\text{block\_width } (\Gamma\_tm \ M)) \ p \leq \text{pos}$ 
 $\wedge \text{pos} < \text{sim\_pos } (\text{block\_width } (\Gamma\_tm \ M)) \ p + \text{block\_width}$ 
 $(\Gamma\_tm \ M)$ 
 $\text{then } \text{write\_bit } (\Gamma\_tm \ M) (\text{bl\_tm } M) (\text{buf } tk)$ 
 $(\text{pos} - \text{sim\_pos } (\text{block\_width } (\Gamma\_tm \ M)) \ p)$ 
 $\text{else } \text{mt\_tape } c' \ tk \ \text{pos}))$ 
 $\wedge \text{mt\_pos } c'' = \text{mt\_pos } c'$ 
  <proof>

```

The write-phase prefix walk: from the write boundary (current-tape field $k_unidx \ 0$, bit-counter 0 , the head positions the read phase left), iterate the unified non-last per-tape write *ar_write_tape_step* over the first j tapes ($j \leq k_tm \ M - 1$, all non-last), landing back at *AR_SimWrite* on tape $k_unidx \ j$. Dual to *ar_read_prefix*: the read kept the tape constant and varied *buf/posk*; the write keeps *buf/posk/positions* constant and varies the *tape*. The descriptor splits on $k_idx \ k < j$: visited tapes carry their over-written block (*?out*, the *write_bit* image for proper tapes, or the unchanged tape for *LE* tapes); the rest are unchanged. Each per-tape step's *tcrr* / position entry facts hold because the current tape is not yet visited ($\text{mt_tape } c \ k_unidx \ j = \text{mt_tape } c0 \ (k_unidx \ j)$). Variable per-tape cost (1 or $2b$), aggregate bounded by $j \cdot 2b$.

Relation-generic write prefix walk scaffold. The single per-tape helper

(*ar_write_tape_step*) is the *leaf* hypothesis, quantified over the per-tape config, tape index, tape-content function, and position; union and sub versions are thin instantiations.

lemma *ar_write_prefix_gen*:

fixes $M :: ('q, 'a) \text{mttm}$
and $cM :: ('a, 'q) \text{mt_config}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$
and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config rel}$
assumes *leaf*:
 $\bigwedge cc \text{ tk}' \text{ tM}' p'.$
 $\llbracket \text{valid_mttm } M; q \in Q_tm \text{ } M; 2 \leq \text{block_width } (\Gamma_tm \text{ } M);$
 $\neg \text{is_last_k } M \text{ tk}';$
 $\text{mt_state } cc = (q, \text{AR_SimWrite}, \text{tk}', 0, \text{buf}, \text{dvec}, \text{posk});$
 $\text{ar_tape_correspondence } (\Gamma_tm \text{ } M) (\text{le_tm } M) (\text{bl_tm } M)$
 $\text{tM}' (\text{mt_tape } cc \text{ tk}');$
 $\text{mt_pos } cc \text{ tk}' = (\text{if } p' = 0 \text{ then } \text{Suc } 0$
 $\text{else } \text{sim_pos } (\text{block_width } (\Gamma_tm \text{ } M)) \text{ } p' + \text{block_width } (\Gamma_tm \text{ } M));$
 $\text{posk } \text{tk}' = \text{AR_AtLE} \longleftrightarrow p' = 0;$
 $\forall k. \text{buf } k \in \Gamma_tm \text{ } M \cup \{\text{bl_tm } M\};$
 $\text{ar_valid_stage } (\Gamma_tm \text{ } M) (\text{bl_tm } M)$
 $(\text{AR_SimWrite}, \text{tk}', 0, \text{buf}, \text{dvec}, \text{posk});$
 $\forall j \geq k_tm \text{ } M. \text{mt_tape } cc \text{ } j (\text{mt_pos } cc \text{ } j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \text{ } M)$
 $(\text{AR_SimWrite}, \text{tk}', 0, \text{buf}, \text{dvec}, \text{posk}) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R' \wedge (\text{if } p' = 0 \text{ then } 1 \text{ else } 2 * \text{block_width } (\Gamma_tm$
 $M))$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimWrite}, k_succ \text{ tk}', 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = (\text{if } p' = 0 \text{ then } \text{mt_tape } cc$
 $\text{else } (\text{mt_tape } cc)(\text{tk}' := (\lambda pos.$
 $\text{if } \text{sim_pos } (\text{block_width } (\Gamma_tm \text{ } M)) \text{ } p' \leq \text{pos}$
 $\wedge \text{pos} < \text{sim_pos } (\text{block_width } (\Gamma_tm \text{ } M)) \text{ } p' + \text{block_width}$
 $(\Gamma_tm \text{ } M)$
 $\text{then } \text{write_bit } (\Gamma_tm \text{ } M) (\text{bl_tm } M) (\text{buf } \text{tk}')$
 $(\text{pos} - \text{sim_pos } (\text{block_width } (\Gamma_tm \text{ } M)) \text{ } p')$
 $\text{else } \text{mt_tape } cc \text{ tk}' \text{ pos})))$
 $\wedge \text{mt_pos } c'' = \text{mt_pos } cc$
and $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \text{ } M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \text{ } M)$
and $stg0: \text{mt_state } c0 = (q, \text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
and $tcorr: \forall k < k_tm \text{ } M. \text{ar_tape_correspondence } (\Gamma_tm \text{ } M) (\text{le_tm } M)$
 $(\text{bl_tm } M)$
 $(\text{mt_tape } cM \text{ } k) (\text{mt_tape } c0 \text{ } k)$
and $ppos: \forall k < k_tm \text{ } M. \text{mt_pos } c0 \text{ } k = (\text{if } \text{mt_pos } cM \text{ } k = 0 \text{ then } \text{Suc } 0$
 $\text{else } \text{sim_pos } (\text{block_width } (\Gamma_tm \text{ } M)) (\text{mt_pos } cM \text{ } k)$
 $+ \text{block_width } (\Gamma_tm \text{ } M))$
and $poskle: \forall k < k_tm \text{ } M. \text{posk } k = \text{AR_AtLE} \longleftrightarrow \text{mt_pos } cM \text{ } k = 0$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm \text{ } M \cup \{\text{bl_tm } M\}$
and $\text{pad0}: \forall j \geq k_tm \text{ } M. \text{mt_tape } c0 \text{ } j (\text{mt_pos } c0 \text{ } j) = \text{BLANK4}$

and src0 : $\text{ar_stage_bounded } (\text{bl_tm } M) (k_tm M)$
 $(\text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $j \leq k_tm M - 1 \implies$
 $(\exists c m. (c0, c) \in R' \wedge m$
 $\wedge m \leq j * (2 * \text{block_width } (\Gamma_tm M))$
 $\wedge \text{mt_state } c = (q, \text{AR_SimWrite}, k_unidx j, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c = (\lambda k. \text{if } k_idx k < j$
 $\text{then } (\text{if } \text{mt_pos } cM k = 0 \text{ then } \text{mt_tape } c0 k$
 $\text{else } (\lambda \text{pos}. \text{if } \text{sim_pos } (\text{block_width } (\Gamma_tm M)) (\text{mt_pos } cM k)$
 $\leq \text{pos}$
 $\wedge \text{pos} < \text{sim_pos } (\text{block_width } (\Gamma_tm M)) (\text{mt_pos } cM k)$
 $+ \text{block_width } (\Gamma_tm M)$
 $\text{then } \text{write_bit } (\Gamma_tm M) (\text{bl_tm } M) (\text{buf } k)$
 $(\text{pos} - \text{sim_pos } (\text{block_width } (\Gamma_tm M)) (\text{mt_pos } cM k))$
 $\text{else } \text{mt_tape } c0 k \text{ pos}))$
 $\text{else } \text{mt_tape } c0 k)$
 $\wedge \text{mt_pos } c = \text{mt_pos } c0)$
 $\langle \text{proof} \rangle$

lemma ar_write_prefix :
fixes $M :: ('q, 'a) \text{mttm}$
and $cM :: ('a, 'q) \text{mt_config}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes vM : $\text{valid_mttm } M$
and qQ : $q \in Q_tm M$
and $kge2$: $2 \leq \text{block_width } (\Gamma_tm M)$
and stg0 : $\text{mt_state } c0 = (q, \text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
and tcrr : $\forall k < k_tm M. \text{ar_tape_correspondence } (\Gamma_tm M) (\text{le_tm } M)$
 $(\text{bl_tm } M)$
 $(\text{mt_tape } cM k) (\text{mt_tape } c0 k)$
and ppos : $\forall k < k_tm M. \text{mt_pos } c0 k = (\text{if } \text{mt_pos } cM k = 0 \text{ then } \text{Suc } 0$
 $\text{else } \text{sim_pos } (\text{block_width } (\Gamma_tm M)) (\text{mt_pos } cM k)$
 $+ \text{block_width } (\Gamma_tm M))$
and poskle : $\forall k < k_tm M. \text{posk } k = \text{AR_AtLE} \longleftrightarrow \text{mt_pos } cM k = 0$
and buf_valid : $\forall k. \text{buf } k \in \Gamma_tm M \cup \{\text{bl_tm } M\}$
and pad0 : $\forall j \geq k_tm M. \text{mt_tape } c0 j (\text{mt_pos } c0 j) = \text{BLANK4}$
and src0 : $\text{ar_stage_bounded } (\text{bl_tm } M) (k_tm M)$
 $(\text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $j \leq k_tm M - 1 \implies$
 $(\exists c m. (c0, c) \in (\text{mttm_step } (\text{alphabet_reduce_delta } M)) \wedge m$
 $\wedge m \leq j * (2 * \text{block_width } (\Gamma_tm M))$
 $\wedge \text{mt_state } c = (q, \text{AR_SimWrite}, k_unidx j, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c = (\lambda k. \text{if } k_idx k < j$
 $\text{then } (\text{if } \text{mt_pos } cM k = 0 \text{ then } \text{mt_tape } c0 k$
 $\text{else } (\lambda \text{pos}. \text{if } \text{sim_pos } (\text{block_width } (\Gamma_tm M)) (\text{mt_pos } cM k)$
 $\leq \text{pos}$
 $\wedge \text{pos} < \text{sim_pos } (\text{block_width } (\Gamma_tm M)) (\text{mt_pos } cM k)$

$cM\ k)$
 $\quad + \text{block_width } (\Gamma_tm\ M)$
 $\quad \text{then write_bit } (\Gamma_tm\ M) (bl_tm\ M) (buf\ k)$
 $\quad \quad (pos - sim_pos (block_width (\Gamma_tm\ M))) (mt_pos$
 $cM\ k))$
 $\quad \quad \text{else mt_tape } c0\ k\ pos))$
 $\quad \text{else mt_tape } c0\ k)$
 $\quad \wedge mt_pos\ c = mt_pos\ c0)$
 $\langle proof \rangle$

The full write phase: the prefix walk over the first $k_tm\ M - 1$ tapes followed by the unified last-tape write *ar_write_tape_finish_step*, landing at *AR_SimAdvance* (current-tape field $k_unidx\ 0$) with every proper tape's b -cell block overwritten by *write_bit* ($buf\ k$) (the encoded M -write symbol) and every *LE* tape and head position unchanged. Aggregate cost $\leq k_tm\ M \cdot 2b$. The split tape descriptor collapses to the full per-tape overwrite on the last tape (every other tape has index $< k_tm\ M - 1$).

Relation-generic full write phase scaffold. The two helpers (prefix walk, last-tape finish) are the *leaf_prefix* and *leaf_finish* hypotheses; union and sub versions are thin instantiations.

lemma *ar_write_phase_gen*:

fixes $M :: ('q, 'a) mttm$
and $cM :: ('a, 'q) mt_config$
and $c0 :: (sym4, 'q \times 'a\ ar_stage) mt_config$
and $R' :: (sym4, 'q \times 'a\ ar_stage) mt_config\ rel$
assumes *leaf_prefix*:
 $\wedge jj. \llbracket \text{valid_mttm } M; q \in Q_tm\ M; 2 \leq block_width (\Gamma_tm\ M);$
 $\quad mt_state\ c0 = (q, AR_SimWrite, 0, 0, buf, dvec, posk);$
 $\quad \forall k < k_tm\ M. ar_tape_correspondence (\Gamma_tm\ M) (le_tm\ M)$
 $(bl_tm\ M)$
 $\quad (mt_tape\ cM\ k) (mt_tape\ c0\ k);$
 $\quad \forall k < k_tm\ M. mt_pos\ c0\ k = (if\ mt_pos\ cM\ k = 0\ then\ Suc\ 0$
 $\quad \quad \text{else } sim_pos (block_width (\Gamma_tm\ M)) (mt_pos\ cM\ k)$
 $\quad \quad + block_width (\Gamma_tm\ M));$
 $\quad \forall k < k_tm\ M. posk\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0;$
 $\quad \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$
 $\quad \forall j \geq k_tm\ M. mt_tape\ c0\ j (mt_pos\ c0\ j) = BLANK4;$
 $\quad ar_stage_bounded (bl_tm\ M) (k_tm\ M)$
 $\quad (AR_SimWrite, 0, 0, buf, dvec, posk);$
 $\quad jj \leq k_tm\ M - 1 \rrbracket$
 $\implies (\exists c\ m. (c0, c) \in R' \wedge \wedge m$
 $\quad \wedge m \leq jj * (2 * block_width (\Gamma_tm\ M))$
 $\quad \wedge mt_state\ c = (q, AR_SimWrite, k_unidx\ jj, 0, buf, dvec, posk)$
 $\quad \wedge mt_tape\ c = (\lambda k. if\ k_idx\ k < jj$
 $\quad \quad \text{then } (if\ mt_pos\ cM\ k = 0\ then\ mt_tape\ c0\ k$
 $\quad \quad \text{else } (\lambda pos. if\ sim_pos (block_width (\Gamma_tm\ M)) (mt_pos$
 $cM\ k) \leq pos$
 $\quad \quad \wedge pos < sim_pos (block_width (\Gamma_tm\ M)))$

$(mt_pos\ cM\ k)$
 $\quad +\ block_width\ (\Gamma_tm\ M)$
 $\quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ k)$
 $\quad (pos - sim_pos\ (block_width\ (\Gamma_tm\ M)))$
 $(mt_pos\ cM\ k))$
 $\quad else\ mt_tape\ c0\ k\ pos))$
 $\quad else\ mt_tape\ c0\ k)$
 $\quad \wedge\ mt_pos\ c = mt_pos\ c0)$
and *leaf_finish*:
 $\wedge_{cc\ tk'\ tM'\ p'}$
 $\quad \llbracket\ valid_mttm\ M; q \in Q_tm\ M; 2 \leq block_width\ (\Gamma_tm\ M);$
 $\quad is_last_k\ M\ tk';$
 $\quad mt_state\ cc = (q, AR_SimWrite, tk', 0, buf, dvec, posk);$
 $\quad ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $\quad \quad tM'\ (mt_tape\ cc\ tk');$
 $\quad mt_pos\ cc\ tk' = (if\ p' = 0\ then\ Suc\ 0$
 $\quad \quad else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p' + block_width\ (\Gamma_tm\ M));$
 $\quad posk\ tk' = AR_AtLE \longleftrightarrow p' = 0;$
 $\quad \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$
 $\quad ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad \quad (AR_SimWrite, tk', 0, buf, dvec, posk);$
 $\quad \forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $\quad ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $\quad \quad (AR_SimWrite, tk', 0, buf, dvec, posk) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R' \wedge \wedge (if\ p' = 0\ then\ 1\ else\ 2 * block_width\ (\Gamma_tm$
 $M))$
 $\quad \wedge\ mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec,$
 $posk)$
 $\quad \wedge\ mt_tape\ c'' = (if\ p' = 0\ then\ mt_tape\ cc$
 $\quad \quad else\ (mt_tape\ cc)(tk' := (\lambda pos.$
 $\quad \quad \quad if\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p' \leq pos$
 $\quad \quad \quad \wedge\ pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ p' + block_width$
 $(\Gamma_tm\ M)$
 $\quad \quad \quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk')$
 $\quad \quad \quad \quad (pos - sim_pos\ (block_width\ (\Gamma_tm\ M))\ p')$
 $\quad \quad \quad \quad else\ mt_tape\ cc\ tk'\ pos)))$
 $\quad \wedge\ mt_pos\ c'' = mt_pos\ cc$
and *vM*: *valid_mttm* *M*
and *qQ*: $q \in Q_tm\ M$
and *kge2*: $2 \leq block_width\ (\Gamma_tm\ M)$
and *stg0*: $mt_state\ c0 = (q, AR_SimWrite, 0, 0, buf, dvec, posk)$
and *tcrrr*: $\forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)$
 $(bl_tm\ M)$
 $\quad (mt_tape\ cM\ k)\ (mt_tape\ c0\ k)$
and *ppos*: $\forall k < k_tm\ M. mt_pos\ c0\ k = (if\ mt_pos\ cM\ k = 0\ then\ Suc\ 0$
 $\quad else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
 $\quad +\ block_width\ (\Gamma_tm\ M))$
and *poskle*: $\forall k < k_tm\ M. posk\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0$
and *buf_valid*: $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$

and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, 0, 0, buf, dvec, posk)$
shows $\exists c\ m. (c0, c) \in R' \wedge m$
 $\wedge m \leq k_tm\ M * (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c = (\lambda k. \text{if } k < k_tm\ M$
 $\text{then } (if\ mt_pos\ cM\ k = 0\ \text{then } mt_tape\ c0\ k$
 $\text{else } (\lambda pos. \text{if } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
 $\leq pos$
 $\wedge pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos$
 $cM\ k)$
 $+ block_width\ (\Gamma_tm\ M)$
 $\text{then } write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ k)$
 $(pos - sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos$
 $cM\ k))$
 $\text{else } mt_tape\ c0\ k\ pos))$
 $\text{else } mt_tape\ c0\ k)$
 $\wedge mt_pos\ c = mt_pos\ c0$
 $\langle proof \rangle$

lemma $ar_write_phase:$

fixes $M :: ('q, 'a)\ mttm$
and $cM :: ('a, 'q)\ mt_config$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0: mt_state\ c0 = (q, AR_SimWrite, 0, 0, buf, dvec, posk)$
and $tcrr: \forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)$
 $(bl_tm\ M)$
 $(mt_tape\ cM\ k)\ (mt_tape\ c0\ k)$
and $ppos: \forall k < k_tm\ M. mt_pos\ c0\ k = (if\ mt_pos\ cM\ k = 0\ \text{then } Suc\ 0$
 $\text{else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
 $+ block_width\ (\Gamma_tm\ M))$
and $poskle: \forall k < k_tm\ M. posk\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, 0, 0, buf, dvec, posk)$
shows $\exists c\ m. (c0, c) \in (mttm_step\ (alphabet_reduce_delta\ M)) \wedge m$
 $\wedge m \leq k_tm\ M * (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c = (\lambda k. \text{if } k < k_tm\ M$
 $\text{then } (if\ mt_pos\ cM\ k = 0\ \text{then } mt_tape\ c0\ k$
 $\text{else } (\lambda pos. \text{if } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
 $\leq pos$
 $\wedge pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos$
 $cM\ k)$

```

      + block_width (Γ_tm M)
      then write_bit (Γ_tm M) (bl_tm M) (buf k)
        (pos - sim_pos (block_width (Γ_tm M))) (mt_pos
cM k))
      else mt_tape c0 k pos))
    else mt_tape c0 k)
  ∧ mt_pos c = mt_pos c0
  ⟨proof⟩

end
theory AlphabetReduction_ForwardAdvance
  imports AlphabetReduction_ForwardWrite
begin

```

2.15 Advance phase

Relation-generic advance left-walk loop scaffold. The chain over R' is built natively by *relpow_invariant_chain* from the per-step leaf contract supplied as the *leaf* hypothesis; the union-relation *ar_advance_walk_loop* and the sub-relation *ar_advance_walk_loop_in_sub* are both thin instantiations, discharging *leaf* by *ar_advance_walk_step* and *ar_advance_walk_step_in_sub* respectively. No *stays* side-condition arises: the chain is native to R' rather than lifted from the union relation, so the leaf-parametric scaffold applies where a union-to-sub chain lift would not. Throughout, b abbreviates *block_width* Γ (the per-symbol cell width).

lemma *ar_advance_walk_loop_gen*:

```

  fixes M :: ('q, 'a) mttm
  and c0 :: (sym4, 'q × 'a ar_stage) mt_config
  and R' :: (sym4, 'q × 'a ar_stage) mt_config rel
  assumes leaf:
    ∧ cc i. [ [ valid_mttm M;
      mt_state cc = (q, AR_SimAdvance, tk, i, buf, dvec, posk);
      q ∈ Q_tm M;
      mt_tape cc tk (mt_pos cc tk) ≠ LE4;
      Suc i < ar_disp (block_width (Γ_tm M)) (dvec tk) (posk tk);
      ar_valid_stage (Γ_tm M) (bl_tm M)
        (AR_SimAdvance, tk, i, buf, dvec, posk);
      ∀ j ≥ k_tm M. mt_tape cc j (mt_pos cc j) = BLANK4;
      ar_stage_bounded (bl_tm M) (k_tm M)
        (AR_SimAdvance, tk, i, buf, dvec, posk) ] ]
    ⇒ ∃ c''. (cc, c'') ∈ R'
      ∧ mt_state c'' = (q, AR_SimAdvance, tk, Suc i, buf, dvec,
posk)
      ∧ mt_tape c'' = mt_tape cc
      ∧ mt_pos c'' = (mt_pos cc)(tk := mt_pos cc tk - 1)
  and vM: valid_mttm M
  and qQ: q ∈ Q_tm M
  and stg: mt_state c0 = (q, AR_SimAdvance, tk, 0, buf, dvec, posk)

```

and *buf_valid*: $\forall k. \text{buf } k \in \Gamma_tm\ M \cup \{\text{bl_tm } M\}$
and *ndisp*: $n < \text{ar_disp } (\text{block_width } (\Gamma_tm\ M))\ (\text{dvec } tk)\ (\text{posk } tk)$
and *notLE*: $\bigwedge j. j < n \implies \text{mt_tape } c0\ tk\ (\text{mt_pos } c0\ tk - j) \neq LE4$
and *pad0*: $\forall j \geq k_tm\ M. \text{mt_tape } c0\ j\ (\text{mt_pos } c0\ j) = BLANK4$
and *src0*: $\text{ar_stage_bounded } (\text{bl_tm } M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c'. (c0, c') \in R' \wedge n$
 $\wedge \text{mt_state } c' = (q, AR_SimAdvance, tk, n, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c'\ tk = \text{mt_pos } c0\ tk - n$
 $\wedge \text{mt_tape } c' = \text{mt_tape } c0$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c'\ k' = \text{mt_pos } c0\ k')$
 $\langle \text{proof} \rangle$

The advance left-walk loop, proper tape. From an *AR_SimAdvance* stage at bit-counter *0* with the head at *start*, *n* *M*'-steps walk the head *n* cells *L* (counter *0* $\rightarrow$ *n*), the tape and all other heads unchanged. Each *ar_advance_walk_step*'s $\neq LE4$ guard reads the current cell *start* $- j$, supplied $\neq LE4$ by *notLE*; the per-step displacement bound *Suc j* $< D$ follows from $n < D$ by *linarith*. Unlike the write loops this loop does not mutate the tape (advance only moves heads), so the invariant keeps *mt_tape c* = *mt_tape c0* constant. Chain length *n*.

lemma *ar_advance_walk_loop*:
fixes *M* :: ('q, 'a) *mttm*
and *c0* :: (*sym4*, 'q $\times$ 'a *ar_stage*) *mt_config*
assumes *vM*: *valid_mttm M*
and *qQ*: $q \in Q_tm\ M$
and *stg*: $\text{mt_state } c0 = (q, AR_SimAdvance, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and *buf_valid*: $\forall k. \text{buf } k \in \Gamma_tm\ M \cup \{\text{bl_tm } M\}$
and *ndisp*: $n < \text{ar_disp } (\text{block_width } (\Gamma_tm\ M))\ (\text{dvec } tk)\ (\text{posk } tk)$
and *notLE*: $\bigwedge j. j < n \implies \text{mt_tape } c0\ tk\ (\text{mt_pos } c0\ tk - j) \neq LE4$
and *pad0*: $\forall j \geq k_tm\ M. \text{mt_tape } c0\ j\ (\text{mt_pos } c0\ j) = BLANK4$
and *src0*: $\text{ar_stage_bounded } (\text{bl_tm } M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c'. (c0, c') \in (\text{mttm_step } (\text{alphabet_reduce_delta } M)) \wedge n$
 $\wedge \text{mt_state } c' = (q, AR_SimAdvance, tk, n, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c'\ tk = \text{mt_pos } c0\ tk - n$
 $\wedge \text{mt_tape } c' = \text{mt_tape } c0$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c'\ k' = \text{mt_pos } c0\ k')$
 $\langle \text{proof} \rangle$

Relation-generic unified single-tape advance scaffold (non-last tape). Both helper references of the dispatch — the walk loop and the boundary step — are abstracted as the *leaf_loop* and *leaf_boundary* hypotheses, so the union-relation *ar_advance_tape_step* and the sub-relation *ar_advance_tape_step_in_sub* are both thin instantiations (discharging the two leaves by the union- resp. sub-relation helper lemmas). Multi-leaf instance of the leaf-parametric scaffold; no *stays* side-condition since both leaves' chains are native to *R'*.

lemma *ar_advance_tape_step_gen*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config rel}$
assumes *leaf_loop*:
 $\bigwedge c0' n. \llbracket \text{valid_mttm } M;$
 $q \in Q_tm \ M;$
 $\text{mt_state } c0' = (q, \text{AR_SimAdvance}, tk, 0, \text{buf}, \text{dvec}, \text{posk});$
 $\forall k. \text{buf } k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\};$
 $n < \text{ar_disp } (\text{block_width } (\Gamma_tm \ M)) (\text{dvec } tk) (\text{posk } tk);$
 $\bigwedge j. j < n \implies \text{mt_tape } c0' \ tk \ (\text{mt_pos } c0' \ tk - j) \neq \text{LE4};$
 $\forall j \geq k_tm \ M. \text{mt_tape } c0' \ j \ (\text{mt_pos } c0' \ j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimAdvance}, tk, 0, \text{buf}, \text{dvec}, \text{posk}) \rrbracket$
 $\implies \exists c'. (c0', c') \in R' \wedge n$
 $\wedge \text{mt_state } c' = (q, \text{AR_SimAdvance}, tk, n, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c' \ tk = \text{mt_pos } c0' \ tk - n$
 $\wedge \text{mt_tape } c' = \text{mt_tape } c0'$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c' \ k' = \text{mt_pos } c0' \ k')$
and *leaf_boundary*:
 $\bigwedge cc \ i. \llbracket \text{valid_mttm } M;$
 $\text{mt_state } cc = (q, \text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk});$
 $q \in Q_tm \ M;$
 $\neg \text{is_last_k } M \ tk;$
 $(\text{dvec } tk = \text{dir}.R \wedge i = 0)$
 $\vee (\text{dvec } tk \neq \text{dir}.R$
 $\wedge \text{mt_tape } cc \ tk \ (\text{mt_pos } cc \ tk) \neq \text{LE4}$
 $\wedge \text{Suc } i = \text{ar_disp } (\text{block_width } (\Gamma_tm \ M)) (\text{dvec } tk) (\text{posk}$
 $tk));$
 $\text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk});$
 $\forall j \geq k_tm \ M. \text{mt_tape } cc \ j \ (\text{mt_pos } cc \ j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimAdvance}, tk, i, \text{buf}, \text{dvec}, \text{posk}) \rrbracket$
 $\implies \exists c''. (cc, c'') \in R'$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimAdvance}, k_succ \ tk, 0, \text{buf},$
 $\text{dvec},$
 $\text{posk}(tk := \text{ar_newpos } (\text{dvec } tk) (\text{posk } tk)))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } cc$
 $\wedge \text{mt_pos } c'' = (\text{if } \text{dvec } tk = \text{dir}.R \text{ then } \text{mt_pos } cc$
 $\text{else } (\text{mt_pos } cc)(tk := \text{mt_pos } cc \ tk - 1))$
and $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{notlast}: \neg \text{is_last_k } M \ tk$
and $\text{stg}: \text{mt_state } c0 = (q, \text{AR_SimAdvance}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\}$
and $\text{dge1}: \text{dvec } tk \neq \text{dir}.R$
 $\implies 0 < \text{ar_disp } (\text{block_width } (\Gamma_tm \ M)) (\text{dvec } tk) (\text{posk } tk)$

and *notLE*: $\bigwedge m. m < ar_disp (block_width (\Gamma_tm M)) (dvec tk) (posk tk)$
 $\implies mt_tape\ c0\ tk\ (mt_pos\ c0\ tk - m) \neq LE4$
and *pad0*: $\forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and *src0*: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in R'$
 $\wedge \wedge (if\ dvec\ tk = dir.R\ then\ 1$
 $\quad else\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk))$
 $\wedge\ mt_state\ c' = (q, AR_SimAdvance, k_succ\ tk, 0, buf, dvec,$
 $\quad posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk)))$
 $\wedge\ mt_tape\ c' = mt_tape\ c0$
 $\wedge\ mt_pos\ c' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c0$
 $\quad else\ (mt_pos\ c0)(tk := mt_pos\ c0\ tk$
 $\quad - ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk$
 $tk)))$
 $\langle proof \rangle$

The unified single-tape advance, non-last tape: the *R*/non-*R* dispatch, the advance analogue of *ar_write_tape_step*. From an *AR_SimAdvance* stage at bit-counter *0*, tape *tk*'s head is repositioned to *M*'s new cell and the hand-off goes to the next tape *k_succ tk* (counter *0*, *posk tk* updated by *ar_newpos*). An *R*-move needs no head motion (the head already sits at *sim_pos (p + 1)*): a single boundary step, cost *1*. A non-*R*-move walks the head *D = ar_disp* cells *L*: the *D - 1*-step *ar_advance_walk_loop* followed by the final boundary *L*-move, cost *D*, landing the head at *start - D*. The *dge1* hypothesis rules out the forbidden *L*-from-*AR_AtLE* (zero displacement) so that *D ≥ 1*; *notLE* covers every cell the walk reads (*start - m*, *m < D*). The tape is unchanged; only the head on *tk* moves.

lemma *ar_advance_tape_step*:

fixes *M* :: ('q, 'a) *mttm*
and *c0* :: (*sym4*, 'q × 'a *ar_stage*) *mt_config*
assumes *vM*: *valid_mttm M*
and *qQ*: *q* ∈ *Q_tm M*
and *kge2*: $2 \leq block_width\ (\Gamma_tm\ M)$
and *notlast*: $\neg is_last_k\ M\ tk$
and *stg*: $mt_state\ c0 = (q, AR_SimAdvance, tk, 0, buf, dvec, posk)$
and *buf_valid*: $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and *dge1*: $dvec\ tk \neq dir.R$
 $\implies 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$
and *notLE*: $\bigwedge m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$
 $\implies mt_tape\ c0\ tk\ (mt_pos\ c0\ tk - m) \neq LE4$
and *pad0*: $\forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and *src0*: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in (mttm_step\ (alphabet_reduce_delta\ M))$
 $\wedge \wedge (if\ dvec\ tk = dir.R\ then\ 1$
 $\quad else\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk))$
 $\wedge\ mt_state\ c' = (q, AR_SimAdvance, k_succ\ tk, 0, buf, dvec,$

$$\begin{aligned}
& \text{posk}(tk := ar_newpos (dvec tk) (posk tk))) \\
& \wedge mt_tape\ c' = mt_tape\ c0 \\
& \wedge mt_pos\ c' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c0 \\
& \quad \text{else}\ (mt_pos\ c0)(tk := mt_pos\ c0\ tk \\
& \quad \quad - ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk \\
& tk))) \\
& \langle proof \rangle
\end{aligned}$$

Relation-generic unified single-tape advance scaffold (last tape). As *ar_advance_tape_step_gen* but the boundary leaf transitions to *AR_SimNext*; both helper references (walk loop, finish step) are the *leaf_loop* and *leaf_finish* hypotheses, so the union- and sub-relation versions are thin instantiations.

lemma *ar_advance_tape_finish_step_gen*:

fixes $M :: ('q, 'a)\ mttm$

and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$

and $R' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config\ rel$

assumes *leaf_loop*:

$\wedge c0'\ n. \llbracket valid_mttm\ M;$

$q \in Q_tm\ M;$

$mt_state\ c0' = (q, AR_SimAdvance, tk, 0, buf, dvec, posk);$

$\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$

$n < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk);$

$\bigwedge j. j < n \implies mt_tape\ c0'\ tk\ (mt_pos\ c0'\ tk - j) \neq LE4;$

$\forall j \geq k_tm\ M. mt_tape\ c0'\ j\ (mt_pos\ c0'\ j) = BLANK4;$

$ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimAdvance, tk, 0, buf, dvec, posk) \rrbracket$

$\implies \exists c'. (c0', c') \in R' \wedge n$

$\wedge mt_state\ c' = (q, AR_SimAdvance, tk, n, buf, dvec, posk)$

$\wedge mt_pos\ c'\ tk = mt_pos\ c0'\ tk - n$

$\wedge mt_tape\ c' = mt_tape\ c0'$

$\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c'\ k' = mt_pos\ c0'\ k')$

and *leaf_finish*:

$\wedge cc\ i. \llbracket valid_mttm\ M;$

$mt_state\ cc = (q, AR_SimAdvance, tk, i, buf, dvec, posk);$

$q \in Q_tm\ M;$

$is_last_k\ M\ tk;$

$(dvec\ tk = dir.R \wedge i = 0)$

$\vee (dvec\ tk \neq dir.R$

$\wedge mt_tape\ cc\ tk\ (mt_pos\ cc\ tk) \neq LE4$

$\wedge Suc\ i = ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk$

$tk));$

$ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$

$(AR_SimAdvance, tk, i, buf, dvec, posk);$

$\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$

$ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimAdvance, tk, i, buf, dvec, posk) \rrbracket$

$\implies \exists c''. (cc, c'') \in R'$

$\wedge mt_state\ c'' = (q, AR_SimNext, k_unidx\ 0, 0, buf, dvec,$

$$\begin{aligned} & \text{posk}(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk))) \\ & \wedge mt_tape\ c'' = mt_tape\ cc \\ & \wedge mt_pos\ c'' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ cc \\ & \quad \text{else}\ (mt_pos\ cc)(tk := mt_pos\ cc\ tk - 1)) \\ \text{and } vM: & \text{valid_mttm } M \\ \text{and } qQ: & q \in Q_tm\ M \\ \text{and } kge2: & 2 \leq block_width\ (\Gamma_tm\ M) \\ \text{and } last: & is_last_k\ M\ tk \\ \text{and } stg: & mt_state\ c0 = (q, AR_SimAdvance, tk, 0, buf, dvec, posk) \\ \text{and } buf_valid: & \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\} \\ \text{and } dge1: & dvec\ tk \neq dir.R \\ & \implies 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk) \\ \text{and } notLE: & \bigwedge m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk) \\ & \implies mt_tape\ c0\ tk\ (mt_pos\ c0\ tk - m) \neq LE4 \\ \text{and } pad0: & \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4 \\ \text{and } src0: & ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M) \\ & (AR_SimAdvance, tk, 0, buf, dvec, posk) \\ \text{shows } \exists c'. & (c0, c') \in R' \\ & \quad \wedge \wedge (if\ dvec\ tk = dir.R\ then\ 1 \\ & \quad \quad \text{else}\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)) \\ & \wedge mt_state\ c' = (q, AR_SimNext, k_unidx\ 0, 0, buf, dvec, \\ & \quad \text{posk}(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk))) \\ & \wedge mt_tape\ c' = mt_tape\ c0 \\ & \wedge mt_pos\ c' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c0 \\ & \quad \text{else}\ (mt_pos\ c0)(tk := mt_pos\ c0\ tk \\ & \quad \quad - ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk \\ & \quad \quad tk))) \\ \langle proof \rangle
\end{aligned}$$

The unified single-tape advance, last tape: as $ar_advance_tape_step$ but tk is the last tape, so the boundary step ($ar_advance_finish_step$) transitions to $AR_SimNext$ (current-tape field reset to $k_unidx\ 0$) rather than advancing to $k_succ\ tk$. Same R /non- R dispatch, the same walk-then-boundary decomposition, and the same conditional head conclusion.

lemma $ar_advance_tape_finish_step$:

fixes $M :: ('q, 'a)\ mttm$

and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$

assumes $vM: \text{valid_mttm } M$

and $qQ: q \in Q_tm\ M$

and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$

and $last: is_last_k\ M\ tk$

and $stg: mt_state\ c0 = (q, AR_SimAdvance, tk, 0, buf, dvec, posk)$

and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$

and $dge1: dvec\ tk \neq dir.R$

$\implies 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$

and $notLE: \bigwedge m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$

$\implies mt_tape\ c0\ tk\ (mt_pos\ c0\ tk - m) \neq LE4$

and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$

and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimAdvance, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in (mttm_step (alphabet_reduce_delta M))$
 $\wedge \wedge (if\ dvec\ tk = dir.R\ then\ 1$
 $\quad else\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk))$
 $\wedge\ mt_state\ c' = (q, AR_SimNext, k_unidx\ 0, 0, buf, dvec,$
 $\quad posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk)))$
 $\wedge\ mt_tape\ c' = mt_tape\ c0$
 $\wedge\ mt_pos\ c' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c0$
 $\quad else\ (mt_pos\ c0)(tk := mt_pos\ c0\ tk$
 $\quad -\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk$
 $\quad tk)))$
 $\langle proof \rangle$

Relation-generic advance prefix walk scaffold. The single per-tape helper ($ar_advance_tape_step$) is the $leaf_tape$ hypothesis, quantified over the per-tape config, tape index, and position-kind function; the union- and sub-relation prefixes are thin instantiations.

lemma $ar_advance_prefix_gen$:

fixes $M :: ('q, 'a) mttm$

and $c0 :: (sym4, 'q \times 'a\ ar_stage) mt_config$

and $R' :: (sym4, 'q \times 'a\ ar_stage) mt_config\ rel$

assumes $leaf_tape$:

$\bigwedge_{cc\ tk'\ pk'.$

$\llbracket valid_mttm\ M; q \in Q_tm\ M; 2 \leq block_width\ (\Gamma_tm\ M);$

$\neg is_last_k\ M\ tk';$

$mt_state\ cc = (q, AR_SimAdvance, tk', 0, buf0, dvec, pk');$

$\forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\};$

$dvec\ tk' \neq dir.R$

$\implies 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk')\ (pk'\ tk');$

$\bigwedge m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk')\ (pk'\ tk')$

$\implies mt_tape\ cc\ tk'\ (mt_pos\ cc\ tk' - m) \neq LE4;$

$\forall j \geq k_tm\ M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$

$ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$

$(AR_SimAdvance, tk', 0, buf0, dvec, pk') \rrbracket$

$\implies \exists c'. (cc, c') \in R'$

$\wedge \wedge (if\ dvec\ tk' = dir.R\ then\ 1$

$\quad else\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk')\ (pk'\ tk'))$

$\wedge\ mt_state\ c' = (q, AR_SimAdvance, k_succ\ tk', 0, buf0, dvec,$

$\quad pk'(tk' := ar_newpos\ (dvec\ tk')\ (pk'\ tk'))$

$\wedge\ mt_tape\ c' = mt_tape\ cc$

$\wedge\ mt_pos\ c' = (if\ dvec\ tk' = dir.R\ then\ mt_pos\ cc$

$\quad else\ (mt_pos\ cc)(tk' := mt_pos\ cc\ tk')$

$\quad -\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk')\ (pk'\ tk'))$

and $vM: valid_mttm\ M$

and $qQ: q \in Q_tm\ M$

and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$

and $stg0: mt_state\ c0 = (q, AR_SimAdvance, 0, 0, buf0, dvec, posk0)$

and $buf_valid: \forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$

and $dge1: \forall k < k_tm\ M. dvec\ k \neq dir.R$

$\longrightarrow 0 < ar_disp (block_width (\Gamma_tm M)) (dvec k) (posk0 k)$
and $notLE: \forall k < k_tm M. \forall m. m < ar_disp (block_width (\Gamma_tm M))$
 $(dvec k) (posk0 k)$
 $\longrightarrow mt_tape\ c0\ k\ (mt_pos\ c0\ k - m) \neq LE4$
and $pad0: \forall j \geq k_tm M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded (bl_tm M) (k_tm M)$
 $(AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
shows $j \leq k_tm M - 1 \implies$
 $(\exists c\ m. (c0, c) \in R' \wedge m$
 $\wedge m \leq j * (2 * block_width (\Gamma_tm M))$
 $\wedge mt_state\ c = (q, AR_SimAdvance, k_unidx\ j, 0, buf0, dvec,$
 $(\lambda k. \text{if } k_idx\ k < j \text{ then } ar_newpos\ (dvec\ k)\ (posk0\ k)$
 $\text{else } posk0\ k))$
 $\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. \text{if } k_idx\ k < j$
 $\text{then } (\text{if } dvec\ k = dir.R \text{ then } mt_pos\ c0\ k$
 $\text{else } mt_pos\ c0\ k$
 $\text{--- } ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ k)\ (posk0\ k))$
 $\text{else } mt_pos\ c0\ k))$
 $\langle proof \rangle$

The advance prefix walk: a custom induction on the tape index j ($j \leq k_tm M - 1$, so every tape it touches is non-last), iterating $ar_advance_tape_step$ from the boundary tape $k_unidx\ 0$. Like the read prefix, the tape is constant and the carried state is a $k_idx\ k < j$ split: tapes already advanced ($k_idx\ k < j$) hold their ar_newpos -updated position-kind and their repositioned head, the rest hold boundary values. The per-tape $notLE/dge1$ entry facts hold of the current tape $?tk$ because it is not yet visited ($?pk\ j\ ?tk = posk0\ ?tk, mt_pos\ c\ ?tk = mt_pos\ c0\ ?tk$). Aggregate cost $\leq j \cdot 2b$ (each per-tape move is at most $2b$ cells, $ar_disp_le_2k$).

lemma $ar_advance_prefix:$
fixes $M :: ('q, 'a) mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width (\Gamma_tm M)$
and $stg0: mt_state\ c0 = (q, AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
and $buf_valid: \forall k. buf0\ k \in \Gamma_tm M \cup \{bl_tm M\}$
and $dge1: \forall k < k_tm M. dvec\ k \neq dir.R$
 $\longrightarrow 0 < ar_disp (block_width (\Gamma_tm M)) (dvec k) (posk0 k)$
and $notLE: \forall k < k_tm M. \forall m. m < ar_disp (block_width (\Gamma_tm M))$
 $(dvec k) (posk0 k)$
 $\longrightarrow mt_tape\ c0\ k\ (mt_pos\ c0\ k - m) \neq LE4$
and $pad0: \forall j \geq k_tm M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded (bl_tm M) (k_tm M)$
 $(AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
shows $j \leq k_tm M - 1 \implies$
 $(\exists c\ m. (c0, c) \in (mttm_step\ (alphabet_reduce_delta\ M)) \wedge m$

$\wedge m \leq j * (2 * \text{block_width } (\Gamma_tm M))$
 $\wedge \text{mt_state } c = (q, \text{AR_SimAdvance}, k_unidx j, 0, \text{buf0}, \text{dvec},$
 $\quad (\lambda k. \text{if } k_idx k < j \text{ then } \text{ar_newpos } (\text{dvec } k) (\text{posk0 } k)$
 $\quad \quad \text{else } \text{posk0 } k))$
 $\wedge \text{mt_tape } c = \text{mt_tape } c0$
 $\wedge \text{mt_pos } c = (\lambda k. \text{if } k_idx k < j$
 $\quad \text{then } (\text{if } \text{dvec } k = \text{dir}.R \text{ then } \text{mt_pos } c0 k$
 $\quad \quad \text{else } \text{mt_pos } c0 k$
 $\quad \quad \quad - \text{ar_disp } (\text{block_width } (\Gamma_tm M)) (\text{dvec } k) (\text{posk0 } k))$
 $\quad \text{else } \text{mt_pos } c0 k))$
 $\langle \text{proof} \rangle$

Relation-generic full advance phase scaffold. The two helpers (prefix walk, last-tape finish) are the *leaf_prefix* and *leaf_finish* hypotheses; the union- and sub-relation phases are thin instantiations.

lemma *ar_advance_phase_gen*:

fixes $M :: ('q, 'a) \text{mttm}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$
and $R' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config rel}$
assumes *leaf_prefix*:
 $\wedge jj. \llbracket \text{valid_mttm } M; q \in Q_tm M; 2 \leq \text{block_width } (\Gamma_tm M);$
 $\text{mt_state } c0 = (q, \text{AR_SimAdvance}, 0, 0, \text{buf0}, \text{dvec}, \text{posk0});$
 $\forall k. \text{buf0 } k \in \Gamma_tm M \cup \{\text{bl_tm } M\};$
 $\forall k < k_tm M. \text{dvec } k \neq \text{dir}.R$
 $\quad \rightarrow 0 < \text{ar_disp } (\text{block_width } (\Gamma_tm M)) (\text{dvec } k) (\text{posk0 } k);$
 $\forall k < k_tm M. \forall m. m < \text{ar_disp } (\text{block_width } (\Gamma_tm M)) (\text{dvec } k) (\text{posk0 } k)$
 $\quad \rightarrow \text{mt_tape } c0 k (\text{mt_pos } c0 k - m) \neq \text{LE4};$
 $\forall j \geq k_tm M. \text{mt_tape } c0 j (\text{mt_pos } c0 j) = \text{BLANK4};$
 $\text{ar_stage_bounded } (\text{bl_tm } M) (k_tm M)$
 $(\text{AR_SimAdvance}, 0, 0, \text{buf0}, \text{dvec}, \text{posk0});$
 $jj \leq k_tm M - 1 \rrbracket$
 $\Rightarrow (\exists c m. (c0, c) \in R' \wedge m$
 $\quad \wedge m \leq jj * (2 * \text{block_width } (\Gamma_tm M))$
 $\quad \wedge \text{mt_state } c = (q, \text{AR_SimAdvance}, k_unidx jj, 0, \text{buf0}, \text{dvec},$
 $\quad \quad (\lambda k. \text{if } k_idx k < jj \text{ then } \text{ar_newpos } (\text{dvec } k) (\text{posk0 } k)$
 $\quad \quad \quad \text{else } \text{posk0 } k))$
 $\quad \wedge \text{mt_tape } c = \text{mt_tape } c0$
 $\quad \wedge \text{mt_pos } c = (\lambda k. \text{if } k_idx k < jj$
 $\quad \quad \text{then } (\text{if } \text{dvec } k = \text{dir}.R \text{ then } \text{mt_pos } c0 k$
 $\quad \quad \quad \text{else } \text{mt_pos } c0 k$
 $\quad \quad \quad - \text{ar_disp } (\text{block_width } (\Gamma_tm M)) (\text{dvec } k) (\text{posk0 } k))$
 $\quad \quad \text{else } \text{mt_pos } c0 k))$
 $\text{and } \text{leaf_finish}:$
 $\wedge cc \text{ tk' pk'}. \llbracket \text{valid_mttm } M; q \in Q_tm M; 2 \leq \text{block_width } (\Gamma_tm M);$
 $\text{is_last_k } M \text{ tk'};$
 $\text{mt_state } cc = (q, \text{AR_SimAdvance}, \text{tk'}, 0, \text{buf0}, \text{dvec}, \text{pk'});$

$\forall k. \text{buf0 } k \in \Gamma_tm M \cup \{bl_tm M\};$
 $dvec\ tk' \neq dir.R$
 $\implies 0 < ar_disp\ (block_width\ (\Gamma_tm M))\ (dvec\ tk')\ (pk'\ tk');$
 $\wedge m. m < ar_disp\ (block_width\ (\Gamma_tm M))\ (dvec\ tk')\ (pk'\ tk)$
 $\implies mt_tape\ cc\ tk'\ (mt_pos\ cc\ tk' - m) \neq LE4;$
 $\forall j \geq k_tm M. mt_tape\ cc\ j\ (mt_pos\ cc\ j) = BLANK4;$
 $ar_stage_bounded\ (bl_tm M)\ (k_tm M)$
 $(AR_SimAdvance, tk', 0, buf0, dvec, pk') \parallel$
 $\implies \exists c'. (cc, c') \in R' \wedge (if\ dvec\ tk' = dir.R\ then\ 1$
 $\quad else\ ar_disp\ (block_width\ (\Gamma_tm M))\ (dvec\ tk')\ (pk'\ tk'))$
 $\wedge mt_state\ c' = (q, AR_SimNext, k_unidx\ 0, 0, buf0, dvec,$
 $\quad pk'(tk' := ar_newpos\ (dvec\ tk')\ (pk'\ tk')))$
 $\wedge mt_tape\ c' = mt_tape\ cc$
 $\wedge mt_pos\ c' = (if\ dvec\ tk' = dir.R\ then\ mt_pos\ cc$
 $\quad else\ (mt_pos\ cc)(tk' := mt_pos\ cc\ tk'$
 $\quad - ar_disp\ (block_width\ (\Gamma_tm M))\ (dvec\ tk')\ (pk'\ tk')))$
and vM : $valid_mttm\ M$
and qQ : $q \in \bar{Q}_tm M$
and $kge2$: $2 \leq block_width\ (\Gamma_tm M)$
and $stg0$: $mt_state\ c0 = (q, AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
and buf_valid : $\forall k. \text{buf0 } k \in \Gamma_tm M \cup \{bl_tm M\}$
and $dge1$: $\forall k < k_tm M. dvec\ k \neq dir.R$
 $\implies 0 < ar_disp\ (block_width\ (\Gamma_tm M))\ (dvec\ k)\ (posk0\ k)$
and $notLE$: $\forall k < k_tm M. \forall m. m < ar_disp\ (block_width\ (\Gamma_tm M))$
 $(dvec\ k)\ (posk0\ k)$
 $\implies mt_tape\ c0\ k\ (mt_pos\ c0\ k - m) \neq LE4$
and $pad0$: $\forall j \geq k_tm M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0$: $ar_stage_bounded\ (bl_tm M)\ (k_tm M)$
 $(AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
shows $\exists c\ m. (c0, c) \in R' \wedge m$
 $\wedge m \leq k_tm M * (2 * block_width\ (\Gamma_tm M))$
 $\wedge mt_state\ c = (q, AR_SimNext, k_unidx\ 0, 0, buf0, dvec,$
 $\quad (\lambda k. if\ k < k_tm M\ then\ ar_newpos\ (dvec\ k)\ (posk0\ k)$
 $\quad \quad else\ posk0\ k))$
 $\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. if\ k < k_tm M$
 $\quad then\ (if\ dvec\ k = dir.R\ then\ mt_pos\ c0\ k$
 $\quad \quad else\ mt_pos\ c0\ k - ar_disp\ (block_width\ (\Gamma_tm M))\ (dvec\ k)$
 $\quad \quad (posk0\ k))$
 $\quad \quad else\ mt_pos\ c0\ k)$
 $\langle proof \rangle$

The full advance phase: the prefix walk over the first $k_tm M - 1$ tapes followed by the unified last-tape advance $ar_advance_tape_finish_step$, landing at $AR_SimNext$ (current-tape field $k_unidx\ 0$) with every tape's head moved to M 's new position (R : unchanged; otherwise $- ar_disp$) and every $posk$ updated by ar_newpos . The tape is unchanged throughout (advance only moves heads). Aggregate cost $\leq k_tm M \cdot 2b$. The final $posk$ /position vectors collapse from the $k_idx\ k < ?m1$ split because, on the

last tape, every other tape has index $< k_tm\ M - 1$ (k_idx bijective into $\{..< k_tm\ M\}$).

lemma *ar_advance_phase*:

fixes $M :: ('q, 'a)\ mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0: mt_state\ c0 = (q, AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
and $buf_valid: \forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $dge1: \forall k < k_tm\ M. dvec\ k \neq dir.R$
 $\longrightarrow 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ k)\ (posk0\ k)$
and $notLE: \forall k < k_tm\ M. \forall m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ k)\ (posk0\ k)$
 $\longrightarrow mt_tape\ c0\ k\ (mt_pos\ c0\ k - m) \neq LE4$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
shows $\exists c\ m. (c0, c) \in (mttm_step\ (alphabet_reduce_delta\ M))\ \wedge^{\wedge} m$
 $\wedge m \leq k_tm\ M * (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c = (q, AR_SimNext, k_unidx\ 0, 0, buf0, dvec,$
 $(\lambda k. if\ k < k_tm\ M\ then\ ar_newpos\ (dvec\ k)\ (posk0\ k)$
 $else\ posk0\ k))$
 $\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. if\ k < k_tm\ M$
 $then\ (if\ dvec\ k = dir.R\ then\ mt_pos\ c0\ k$
 $else\ mt_pos\ c0\ k - ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ k)$
 $(posk0\ k))$
 $else\ mt_pos\ c0\ k)$
 $\langle proof \rangle$

end

theory *AlphabetReduction_Forward*

imports *AlphabetReduction_ForwardAdvance*

begin

2.16 Per- M -step simulation phase and the chunked engine

Acceptance read-out, the AR analogue of AE's *ae_simulates_accept_iff*. Under *ar_simulates*, the source M -config is in the accept state iff the paired M' -config sits in the canonical accept config $(t_tm\ M, ar_accept_stage\ bl_M)$. Forward: $q_M = t$ forces the second *ar_simulates* disjunct (the first needs $q_{M'} \notin \{t, r\}$; the third needs $t = r$, ruled out by *valid_mttm_t_neq_r*). Reverse is immediate from the state-equality conjunct. Throughout, b abbreviates *block_width* Γ (the per-symbol cell width).

lemma *ar_simulates_accept_iff*:

fixes $M :: ('q, 'a)\ mttm$

and $cM :: ('a, 'q)\ mt_config$

and $c' :: (\text{sym}4, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{sim}: \text{ar_simulates } M \text{ cM } c'$
shows $(\text{mt_state } cM = t_tm \ M)$
 $\longleftrightarrow (\text{mt_state } c' = (t_tm \ M, \text{ar_accept_stage } (bl_tm \ M)))$
 $\langle \text{proof} \rangle$

The next substep, non-terminal hand-off: a single M' -step from an $AR_SimNext$ stage whose M -state q is neither accepting nor rejecting routes to the next M -step's $AR_SimRead$ boundary, resetting both the current-tape and bit-counter fields to 0 while preserving buf , $dvec$, and the per-tape $posk$. No tape cell changes and no head moves (all directions N , the write vector is the read vector). The cleanest of the five phases — the ar_compute_step shape with arm 1 of the three-arm ar_delta_next union; the two δLE filters discharge reflexively (write equals read, move N) and target-stage validity rests only on $0 < b$. The terminal arms ($q = t_tm \ M / q = r_tm \ M$) route to the halt stages instead and are handled at assembly, not here.

lemma ar_next_step :
fixes $M :: ('q, 'a) \text{ mttm}$
and $c' :: (\text{sym}4, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, AR_SimNext, tk, i, buf, dvec, posk)$
and $qQ: q \in Q_tm \ M$
and $\text{not}T: q \neq t_tm \ M$
and $\text{not}R: q \neq r_tm \ M$
and $\text{vsrce}: \text{ar_valid_stage } (\Gamma_tm \ M) (bl_tm \ M)$
 $(AR_SimNext, tk, i, buf, dvec, posk)$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j (\text{mt_pos } c' \ j) = \text{BLANK}4$
and $\text{src_bounded}: \text{ar_stage_bounded } (bl_tm \ M) (k_tm \ M)$
 $(AR_SimNext, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (q, AR_SimRead, 0, 0, buf, dvec, posk)$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = \text{mt_pos } c'$
 $\langle \text{proof} \rangle$

The two terminal hand-offs, the accept and reject arms of ar_delta_next : a single M' -step from an $AR_SimNext$ stage whose M -state q is the accepting (resp. rejecting) state lands on the canonical halt stage $\text{ar_accept_stage } (bl_tm \ M)$ (resp. ar_reject_stage), so the full target state equals t_tm ($\text{alphabet_reduce } M$) (resp. r_tm): full-state acceptance forces the canonicalisation. Tape and heads are unchanged (the move vector is all- N , no writes). Modelled on ar_next_step ; the membership lets blast select the halt comprehension and instantiate its witnesses.

lemma ar_accept_step :
fixes $M :: ('q, 'a) \text{ mttm}$

and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimNext}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qT: q = t_tm \ M$
and $\text{vsrvc}: \text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimNext}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimNext}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (t_tm \ M, \text{ar_accept_stage } (\text{bl_tm } M))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = \text{mt_pos } c'$
 $\langle \text{proof} \rangle$

lemma ar_reject_step :
fixes $M :: ('q, 'a) \text{ mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimNext}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qR: q = r_tm \ M$
and $\text{vsrvc}: \text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimNext}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimNext}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
 $\wedge \text{mt_state } c'' = (r_tm \ M, \text{ar_reject_stage } (\text{bl_tm } M))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = \text{mt_pos } c'$
 $\langle \text{proof} \rangle$

Two position-bookkeeping facts for the advance phase, decided once in a clean context (no *let*-bound ? -vars to tangle the case split). Given M 's head at position p moving dr (with $dr \neq L$ when $p = 0$, the δLE discipline), the read-phase position-kind *if* $p=0$ *then* *AtLE* *else if* $p=1$ *then* *FirstProper* *else* *FurtherProper* advances via ar_newpos to *AtLE* exactly when M 's new position is 0 , and the read-phase landing position less the ar_disp displacement is exactly sim_pos of M 's new position. Both reduce to a $p \in \{0, 1, \geq 2\}$ cross $dr \in \{N, R, L\}$ grid.

lemma $\text{ar_newpos_atLE_iff}$:
fixes $p :: \text{nat}$ **and** $dr :: \text{dir}$
assumes $nL0: p = 0 \implies dr \neq \text{dir}.L$
shows $(\text{ar_newpos } dr \ (\text{if } p = 0 \text{ then } \text{AR_AtLE}$
 $\text{else if } p = 1 \text{ then } \text{AR_AtFirstProper}$
 $\text{else } \text{AR_AtFurtherProper}) = \text{AR_AtLE})$
 $\longleftrightarrow \text{go_dir } dr \ p = 0$
 $\langle \text{proof} \rangle$

lemma *ar_advance_newsimpos*:
fixes $K\ p :: \text{nat}$ **and** $dr :: \text{dir}$
assumes $Kge2$: $2 \leq K$
and $nL0$: $p = 0 \implies dr \neq \text{dir}.L$
shows $(\text{if } dr = \text{dir}.R$
 $\quad \text{then } (\text{if } p = 0 \text{ then } \text{Suc } 0 \text{ else } \text{sim_pos } K\ p + K)$
 $\quad \text{else } (\text{if } p = 0 \text{ then } \text{Suc } 0 \text{ else } \text{sim_pos } K\ p + K)$
 $\quad \quad - \text{ar_disp } K\ dr\ (\text{if } p = 0 \text{ then } \text{AR_AtLE}$
 $\quad \quad \quad \text{else if } p = 1 \text{ then } \text{AR_AtFirstProper}$
 $\quad \quad \quad \text{else } \text{AR_AtFurtherProper}))$
 $= \text{sim_pos } K\ (\text{go_dir } dr\ p)$
 $\langle \text{proof} \rangle$

Per- M -step forward simulation: one M -step $cM \rightarrow cM_1$ from an AR_SimRead boundary is matched by a bounded M' -phase (read $\rightarrow$ compute $\rightarrow$ write $\rightarrow$ advance $\rightarrow$ next) of at most $k_tm\ M \cdot (5b + 2) + 2$ M' -steps (the per-phase $k_tm\ M$ -scaled sum: read $\leq k_tm\ M \cdot (b+2)$, write and advance each $\leq k_tm\ M \cdot 2b$, compute and next one step apiece) that re-establishes ar_simulates at the next boundary. This is the composition of the per-substep phase lemmas above into one M -step; the chunked engine below iterates it. The reachability hypothesis reach_M with w_sub supplies the substrate LE -only-at-position-0 fact the read look-back needs, and card_ge gives $b \geq 2$ for the proper-region read arms.

The post-write tape correspondence, factored out of $\text{ar_simulates_forward_step}$ so the reverse cycle-close reuses it. Given the read-boundary correspondence between M 's tape and the output tape rt , and that cM_1 is M after writing a' under each head, the write phase's per-tape overwrite wt — which stamps a' 's cell block at the head's block and leaves the rest of rt untouched — is again a correspondence, now for cM_1 . Both arms supply their own wt and discharge wt_def from their write-phase output contract.

lemma *ar_write_tape_correspondence*:
fixes $M :: ('q, 'a)\ \text{mttm}$
and $cM\ cM_1 :: ('a, 'q)\ \text{mt_config}$
and $rt\ wt :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{sym4}$
and $a' :: \text{nat} \Rightarrow 'a$
assumes $kge2$: $2 \leq \text{block_width } (\Gamma_tm\ M)$
and kN : $k < k_tm\ M$
and $tcorr$: $\forall k < k_tm\ M. \text{ar_tape_correspondence } (\Gamma_tm\ M)\ (le_tm\ M)$
 $(bl_tm\ M)$
 $(\text{mt_tape } cM\ k)\ (rt\ k)$
and $cM1\text{tape}$: $\bigwedge k. \text{mt_tape } cM_1\ k = (\text{mt_tape } cM\ k)(\text{mt_pos } cM\ k := a'$
 $k)$
and $a'le0$: $\forall k < k_tm\ M. \text{mt_pos } cM\ k = 0 \longrightarrow a'\ k = le_tm\ M$
and wt_def : $\bigwedge k\ pos. wt\ k\ pos$
 $= (\text{if } \text{mt_pos } cM\ k = 0 \text{ then } rt\ k\ pos$
 $\quad \text{else if } \text{sim_pos } (\text{block_width } (\Gamma_tm\ M))\ (\text{mt_pos } cM\ k) \leq pos$

```

       $\wedge$  pos < sim_pos (block_width ( $\Gamma_{tm}$  M)) (mt_pos cM k)
      + block_width ( $\Gamma_{tm}$  M)
    then write_bit ( $\Gamma_{tm}$  M) (bl_tm M) (a' k)
      (pos - sim_pos (block_width ( $\Gamma_{tm}$  M)) (mt_pos cM k))
    else rt k pos)
  shows ar_tape_correspondence ( $\Gamma_{tm}$  M) (le_tm M) (bl_tm M)
    (mt_tape cM_1 k) (wt k)
<proof>

```

```

lemma ar_simulates_forward_step:
  fixes M :: ('q, 'a) mttm
  and w :: 'a list
  and cM cM_1 :: ('a, 'q) mt_config
  and c' :: (sym4, 'q  $\times$  'a ar_stage) mt_config
  assumes vM: valid_mttm M
  and w_sub: set w  $\subseteq$  Sigma_tm M
  and card_ge: card ( $\Gamma_{tm}$  M)  $\geq$  4
  and reach_M: (init_config_mttm M w, cM)  $\in$  (mttm_step (delta_tm M))*
  and step: (cM, cM_1)  $\in$  mttm_step (delta_tm M)
  and sim: ar_simulates M cM c'
  and posk_ok: ar_posk_consistent M cM c'
  and rbnd: ar_at_read_boundary M c'
  and lebl: le_tm M  $\neq$  bl_tm M
  obtains m c'' where
    m  $\leq$  k_tm M * (5 * block_width ( $\Gamma_{tm}$  M) + 2) + 2
  and (c', c'')  $\in$  mttm_step (alphabet_reduce_delta M) ^^ m
  and ar_simulates M cM_1 c''
  and ar_posk_consistent M cM_1 c''
  and ar_at_read_boundary M c''
<proof>

```

Chunked-induction engine for the simulation phase, the AR analogue of AE's *ae_simulation_phase_chunked*. Given an accepting M -path of length n from a reachable cM with a paired *AR_SimRead* M' -config c' satisfying *ar_simulates*, exhibit a corresponding accepting M' -path of length at most $(5b + 2) \cdot n$ ending in the canonical accept config. Unlike AE — which groups c source steps per stage and needs the strong *less_induct* with *min* n c chunk arithmetic — AR simulates one M -step per phase, so ordinary *induction* n with the fixed per-step bound suffices: each step contributes $\leq 5b + 2$, summing to $(5b + 2) \cdot n$ by a single *linarith*. All substrate reasoning is delegated to *ar_simulates_forward_step*; the engine only splits the trace, recurses on the tail, and composes the two M' -paths via *relpow_add*.

```

lemma ar_simulation_phase_chunked:
  fixes M :: ('q, 'a) mttm
  and w :: 'a list
  and cM cM_final :: ('a, 'q) mt_config
  and c' :: (sym4, 'q  $\times$  'a ar_stage) mt_config
  and n :: nat

```

```

assumes  $vM$ :  $\text{valid\_mttm } M$ 
and  $w\_sub$ :  $\text{set } w \subseteq \text{Sigma\_tm } M$ 
and  $card\_ge$ :  $\text{card } (\Gamma\_tm \ M) \geq 4$ 
and  $reach\_M$ :  $(\text{init\_config\_mttm } M \ w, \ cM) \in (\text{mttm\_step } (\text{delta\_tm } M))^*$ 
and  $trace$ :  $(cM, \ cM\_final) \in (\text{mttm\_step } (\text{delta\_tm } M))^{\wedge n}$ 
and  $accept$ :  $\text{mt\_state } cM\_final = t\_tm \ M$ 
and  $sim$ :  $\text{ar\_simulates } M \ cM \ c'$ 
and  $posk\_ok$ :  $\text{ar\_posk\_consistent } M \ cM \ c'$ 
and  $rbnd$ :  $\text{ar\_at\_read\_boundary } M \ c'$ 
and  $lebl$ :  $\text{le\_tm } M \neq \text{bl\_tm } M$ 
shows  $\exists m \ c''. m \leq (k\_tm \ M$ 
     $\quad * (5 * \text{block\_width } (\Gamma\_tm \ M) + 2) + 2) * n$ 
     $\quad \wedge (c', \ c'') \in (\text{mttm\_step } (\text{alphabet\_reduce\_delta } M))^{\wedge m}$ 
     $\quad \wedge \text{mt\_state } c'' = (t\_tm \ M, \ \text{ar\_accept\_stage } (\text{bl\_tm } M))$ 
 $\langle \text{proof} \rangle$ 

end
theory AlphabetReduction_Theorems
imports AlphabetReduction_Forward
begin

```

2.17 Top-level theorems

Per-substep typing facts: for each substep relation in the five-substep union, the source and target states' $'q$ components lie in $Q_tm \ M$, and the source's ar_substep_idx tag is the relation-specific value (never AR_HaltAccept or AR_HaltReject). Used by $\text{alphabet_reduce_wf}$'s δ' -typing conjunct to show δ' -tuples land in $(Q' - \{\!|t', r'|\!\}) \times \dots \times Q' \times \dots$.

```

lemma  $\text{ar\_delta\_read\_typing}$ :
assumes  $(s, \ a, \ s', \ a', \ d) \in \text{ar\_delta\_read } M$ 
shows  $\text{fst } s \in Q\_tm \ M \wedge \text{fst } (\text{snd } s) = \text{AR\_SimRead}$ 
     $\quad \wedge \text{fst } s' \in Q\_tm \ M$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma  $\text{ar\_delta\_compute\_typing}$ :
assumes  $\text{valM}$ :  $\text{valid\_mttm } M$ 
and  $tr$ :  $(s, \ a, \ s', \ a', \ d) \in \text{ar\_delta\_compute } M$ 
shows  $\text{fst } s \in Q\_tm \ M \wedge \text{fst } (\text{snd } s) = \text{AR\_SimCompute}$ 
     $\quad \wedge \text{fst } s' \in Q\_tm \ M$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma  $\text{ar\_delta\_write\_typing}$ :
assumes  $(s, \ a, \ s', \ a', \ d) \in \text{ar\_delta\_write } M$ 
shows  $\text{fst } s \in Q\_tm \ M \wedge \text{fst } (\text{snd } s) = \text{AR\_SimWrite}$ 
     $\quad \wedge \text{fst } s' \in Q\_tm \ M$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma  $\text{ar\_delta\_advance\_typing}$ :

```

assumes $(s, a, s', a', d) \in ar_delta_advance\ M$
shows $fst\ s \in Q_tm\ M \wedge fst\ (snd\ s) = AR_SimAdvance$
 $\wedge fst\ s' \in Q_tm\ M$
 $\langle proof \rangle$

lemma *ar_delta_next_typing*:
assumes $valM: valid_mttm\ M$
and $tr: (s, a, s', a', d) \in ar_delta_next\ M$
shows $fst\ s \in Q_tm\ M \wedge fst\ (snd\ s) = AR_SimNext$
 $\wedge fst\ s' \in Q_tm\ M$
 $\langle proof \rangle$

State-shape extraction for *alphabet_reduce_delta*: every transition's source and target carry an M -state in Q , and the source stage's index is a *simulation* index — never a halt index — so the source state is distinct from both canonical halt states. Case-split across the five substep relations (the delta is their union intersected with global guards, so membership lands in the union), each branch discharged by its typing lemma; the halt-index distinctness is datatype-level.

lemma *alphabet_reduce_delta_state_shape*:
assumes $valM: valid_mttm\ M$
and $mem: (s, a, s', a', d) \in alphabet_reduce_delta\ M$
shows $fst\ s \in Q_tm\ M \wedge fst\ s' \in Q_tm\ M$
 $\wedge fst\ (snd\ s) \neq AR_HaltAccept$
 $\wedge fst\ (snd\ s) \neq AR_HaltReject$
 $\langle proof \rangle$

Output well-formedness: *alphabet_reduce* preserves the substrate's wf predicate when the input tape alphabet has at least 4 symbols. The reduced machine's tape alphabet is the whole finite type *sym4*, so the read / write codomain conditions are vacuous (*UNIV*), and the δLE -preservation, δLE -no-write, and δ -support-past-tape-count conjuncts are read straight off *alphabet_reduce_delta*'s three intersection guards — no substep-builder unfold. The only structural work is finite Q' (product of finite Q with the bounded valid stages, *finite_ar_valid_stages*), the three distinguished states landing in Q' (their stages are valid and bounded), and the δ -shape's source / target state membership (*alphabet_reduce_delta_state_shape* plus the stage guards).

theorem *alphabet_reduce_wf*:
fixes $M :: ('q, 'a)\ mttm$
assumes $valM: valid_mttm\ M$
and $cardG: card\ (\bar{\Gamma}_tm\ M) \geq 4$
shows $valid_mttm$
 $(alphabet_reduce\ M$
 $:: ('q \times 'a\ ar_stage, sym4)\ mttm)$
 $\langle proof \rangle$

The reduced machine is *le-unique* unconditionally: its δ' is intersected

with the support filter $\{(s, a, s', a', d). \forall k. a' k = LE4 \longrightarrow a k = LE4\}$ (the *le_unique* clause, with *le_tm* (*alphabet_reduce* *M*) = *LE4*), so every produced transition writes *LE4* only where it read *LE4* — no hypothesis on *M* at all. This is the output-side guarantee that makes *alphabet_reduce* a *cut-absorbing* normaliser: even a source machine that has planted a fresh *le* (cut its tape) reduces to a machine whose only *LE4* cell is the mandatory boundary.

lemma *alphabet_reduce_le_unique*:
le_unique (*alphabet_reduce* *M*)
 <proof>

Output well-formedness, the strong form: *alphabet_reduce* maps any valid *M* (with a ≥ 4 -symbol tape alphabet) to a *well-formed* machine — *valid_mttm* plus the three non-degeneracy conditions plus *le_unique*. The start-vs-halt-state distinctness is structural (the *AR_SimRead* init stage differs from the *AR_HaltAccept* / *AR_HaltReject* halt stages in its substep tag), so no $s \neq t$ hypothesis on *M* is needed. This is the precondition the alphabet-enlargement combinator requires of its input, so it is the bridge that lets *alphabet_reduce*'s output feed *alphabet_enlarge* in a composition.

theorem *alphabet_reduce_well_formed*:
 fixes *M* :: ('q, 'a) mttm
 assumes *valM*: *valid_mttm* *M*
 and *cardG*: *card* (Γ_tm *M*) ≥ 4
 shows *well_formed_mttm*
 (*alphabet_reduce* *M* :: ('q $\times$ 'a *ar_stage*, *sym4*) mttm)
 <proof>

Initial-configuration accessors, in terms of the machine's structural projections. Generic (any *M*); they let the init-correspondence proofs read the start tape / state / heads without an inline *cases* *M*.

lemma *mt_tape_init_config*:
mt_tape (*init_config_mttm* *M* *w*)
 = ($\lambda k n. \text{if } k < k_tm \text{ } M$
 then (*if* $n = 0$ *then* *le_tm* *M*
 else if $k = 0 \wedge n \leq \text{length } w$ *then* $w ! (n - 1)$
 else *bl_tm* *M*)
else *bl_tm* *M*)
 <proof>

lemma *mt_state_init_config*:
mt_state (*init_config_mttm* *M* *w*) = *s_tm* *M*
 <proof>

lemma *mt_pos_init_config*:
mt_pos (*init_config_mttm* *M* *w*) = ($\lambda_ . 0$)
 <proof>

Specialised initial tape of the reduced machine: LE_4 at position 0, the (already-encoded) input w' on tape 0, and $BLANK_4$ everywhere else. Stated with w' free and an M -free right-hand side, so it closes by the same *cases* $M/alphabet_reduce_def$ route as the structural projections — sidestepping selector-reduction on $\Gamma_tm\ M$ etc.

lemma $mt_tape_init_config_ar$:
 $mt_tape\ (init_config_mttm\ (alphabet_reduce\ M)\ w')$
 $= (\lambda k\ n.\ \text{if } k < k_tm\ M$
 $\quad \text{then } (\text{if } n = 0 \text{ then } LE_4$
 $\quad \quad \text{else if } k = 0 \wedge n \leq \text{length } w' \text{ then } w'!\ (n - 1)$
 $\quad \quad \text{else } BLANK_4)$
 $\quad \text{else } BLANK_4)$
 $\langle proof \rangle$

Initial-tape correspondence: at the start configuration the encoded input $encode_input_ar\ \Gamma\ bl\ w$ lays out, block by block, exactly as $cell_repr$ demands of M 's initial tape. The proper region (M -position $1 \leq p \leq |w|$) reads off $nth_encode_input_ar$; the blank tail ($p > |w|$) and the non-input tapes ($k \neq 0$) both reduce to the all- $BLANK_4$ block $cell_repr\ \Gamma\ bl\ bl$.

lemma $ar_tape_correspondence_init$:
fixes $M :: ('q, 'a)\ mttm$
assumes vM : $valid_mttm\ M$
and wS : $set\ w \subseteq Sigma_tm\ M$
and kK : $k < k_tm\ M$
shows $ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $(mt_tape\ (init_config_mttm\ M\ w)\ k)$
 $(mt_tape\ (init_config_mttm\ (alphabet_reduce\ M)$
 $\quad (encode_input_ar\ (\Gamma_tm\ M)\ (bl_tm\ M)\ w))\ k)$
 $(is\ ar_tape_correspondence\ ?G\ ?le\ ?bl\ ?tM\ ?tM')$
 $\langle proof \rangle$

The three simulation invariants at the start configuration — the entry premises the chunked engine $ar_simulation_phase_chunked$ consumes. $ar_simulates_init$ carries the weight (its $AR_SimRead$ arm needs $s \neq t$, $s \neq r$ from $valid_mttm$, and its tape clause is $ar_tape_correspondence_init$); the other two read straight off ar_init_stage .

lemma $ar_simulates_init$:
fixes $M :: ('q, 'a)\ mttm$
assumes vM : $valid_mttm\ M$
and wS : $set\ w \subseteq Sigma_tm\ M$
and snt : $s_tm\ M \neq t_tm\ M$
and snr : $s_tm\ M \neq r_tm\ M$
shows $ar_simulates\ M\ (init_config_mttm\ M\ w)$
 $(init_config_mttm\ (alphabet_reduce\ M)$
 $\quad (encode_input_ar\ (\Gamma_tm\ M)\ (bl_tm\ M)\ w))$
 $\langle proof \rangle$

lemma *ar_posk_consistent_init*:
shows *ar_posk_consistent* M (*init_config_mttm* M w)
 (*init_config_mttm* (*alphabet_reduce* M)
 (*encode_input_ar* (Γ_{tm} M) (*bl_tm* M) w))
<proof>

lemma *ar_at_read_boundary_init*:
assumes vM : *valid_mttm* M
shows *ar_at_read_boundary* M
 (*init_config_mttm* (*alphabet_reduce* M)
 (*encode_input_ar* (Γ_{tm} M) (*bl_tm* M) w))
<proof>

The encoded input is a *BIT0/BIT1* string, unconditionally: every block is *encode_symbol*'s image, whose cells lie in $\{\text{BIT0}, \text{BIT1}\}$ by *encode_symbol_cell_domain* — so this needs no $set\ w \subseteq \Sigma$ hypothesis. It discharges the set-containment side of *Lang_mttm* (*alphabet_reduce* M) membership.

lemma *set_encode_input_ar*:
 $set\ (encode_input_ar\ \Gamma\ bl\ w) \subseteq \{\text{BIT0}, \text{BIT1}\}$
<proof>

Forward language preservation: an accepting M -run on w lifts to an accepting M' -run on the encoded input, via the chunked engine off the initial correspondence. This is the forward half of *alphabet_reduce_language*; it mirrors *alphabet_enlarge_language_forward*. The reverse half, and the full biconditional *alphabet_reduce_language*, live in *AlphabetReduction_Reverse.thy* (which imports this theory), supplied by the chunked-reverse engine *ar_simulation_phase_chunked_reverse*.

theorem *alphabet_reduce_language_forward*:
fixes $M :: ('q, 'a)\ mttm$
assumes vM : *valid_mttm* M
 and s_neq_t : $s_tm\ M \neq t_tm\ M$
 and s_neq_r : $s_tm\ M \neq r_tm\ M$
 and le_neq_bl : $le_tm\ M \neq bl_tm\ M$
 and $card_ge$: $card\ (\Gamma_{tm}\ M) \geq 4$
shows $\forall w. set\ w \subseteq Sigma_tm\ M \longrightarrow w \in Lang_mttm\ M$
 $\longrightarrow encode_input_ar\ (\Gamma_{tm}\ M)\ (bl_tm\ M)\ w$
 $\in Lang_mttm\ (alphabet_reduce\ M)$
 $:: ('q \times 'a\ ar_stage, sym4)\ mttm$
<proof>

The per- M -step super-step count $C \cdot (5 \cdot k + 2) + 2$ ($C = k_tm\ M$, $k = b = block_width$) folds into one b -factor $(6 \cdot C + 1) \cdot k$. The slack is $(C + 1) \cdot (k - 2) \geq 0$, tight at $k = 2$ — so $6 \cdot C + 1$ is the least factor that absorbs the count for every $k \geq 2$. The reduction always encodes into blocks of width ≥ 2 (from $card\ \Gamma_{tm}\ M \geq 4$), so the $b \geq 2$ precondition is always met; the looser $b \geq 1$ fold to $7 \cdot C + 2$ is unnecessary.

lemma *ar_time_bound_arith_sharp*:
fixes $C\ k :: \text{nat}$
assumes $2 \leq k$
shows $C * (5 * k + 2) + 2 \leq (6 * C + 1) * k$
<proof>

Time bound under weak time-bounded acceptance, with explicit constants. If M accepts w within $T(|w|)$ steps, the output machine accepts the encoded input within $(6 \cdot k_{tm}\ M + 1) \cdot b \cdot T(|w|)$ steps, where $b = \text{block_width}\ (\Gamma_{tm}\ M)$ is the per-symbol block length (in sym_4 cells) and $k_{tm}\ M$ is the tape count. The exact per- M -step super-step count is $k_{tm}\ M \cdot (5 \cdot b + 2) + 2$, folded up to $(6 \cdot k_{tm}\ M + 1) \cdot b$ using $b \geq 2$ (*ar_time_bound_arith_sharp*; $b \geq 2$ holds because $\text{card}\ (\Gamma_{tm}\ M) \geq 4$, and the factor is tight at $b = 2$, i.e. $\text{card}\ (\Gamma_{tm}\ M) = 4$); the linear-in- $|w|$ coefficient and the additive constant are both 0. b is logarithmic in the source-alphabet cardinality $\text{card}\ (\Gamma_{tm}\ M)$ (the binary-encoding design point) — a *ceiling* log, so the slowdown factor is a step function of $\text{card}\ (\Gamma_{tm}\ M)$ that jumps by one at each power of two (the band $2^{b-1} < \text{card}\ (\Gamma_{tm}\ M) \leq 2^b$ is pinned by *card_le_two_pow_block_width* and *two_pow_block_width_pred_less_card*). The count is the whole alphabet, blank included at index 0 (the endmarker *le* is a genuinely coded symbol, so it too is counted); so b is the uniform-code width, one cell above the coding minimum $\lceil \log_2 (\text{card}\ (\Gamma_{tm}\ M) - 1) \rceil$ just past a power of two — see *block_width*. Weak acceptance, not the strong all-paths *upperb_time_mttm*, is the target: the strong bound is unprovable for the no-validation construction, and weak acceptance serves both the DTM and NDTM applications. Mirrors *alphabet_enlarge_time_explicit*. The classical existential form is the *obtains*-corollary *alphabet_reduce_time* below.

theorem *alphabet_reduce_time_explicit*:
fixes $M :: ('q, 'a)\ \text{mttm}$
and $T :: \text{nat} \Rightarrow \text{nat}$
assumes $vM: \text{valid_mttm}\ M$
and $s_neq_t: s_{tm}\ M \neq t_{tm}\ M$
and $s_neq_r: s_{tm}\ M \neq r_{tm}\ M$
and $le_neq_bl: le_{tm}\ M \neq bl_{tm}\ M$
and $\text{card_ge}: \text{card}\ (\Gamma_{tm}\ M) \geq 4$
shows $\forall w. \text{set}\ w \subseteq \text{Sigma_tm}\ M$
 $\rightarrow \text{accepts_in_time_mttm}\ M\ w\ (T\ (\text{length}\ w))$
 $\rightarrow \text{accepts_in_time_mttm}$
 $\quad (\text{alphabet_reduce}\ M$
 $\quad \quad :: ('q \times 'a\ \text{ar_stage}, \text{sym}_4)\ \text{mttm})$
 $\quad (\text{encode_input_ar}\ (\Gamma_{tm}\ M)\ (bl_{tm}\ M)\ w)$
 $\quad ((6 * k_{tm}\ M + 1) * \text{block_width}\ (\Gamma_{tm}\ M) * T\ (\text{length}\ w))$
<proof>

Linear-time slowdown, classical existential form: reducing the tape alphabet to the fixed type sym_4 multiplies the running time by only a con-

stant times the per-symbol block width. The *obtains*-corollary of *alphabet_reduce_time_explicit* above, hiding its four explicit constants behind existentials instantiated at $d = 6 \cdot k_tm\ M + 1$, $e = f = 0$, and $b = block_width\ (\Gamma_tm\ M)$.

```
theorem alphabet_reduce_time:
  fixes  $M :: ('q, 'a)\ mttm$ 
  and  $T :: nat \Rightarrow nat$ 
  assumes  $vM: valid\_mttm\ M$ 
    and  $s\_neq\_t: s\_tm\ M \neq t\_tm\ M$ 
    and  $s\_neq\_r: s\_tm\ M \neq r\_tm\ M$ 
    and  $le\_neq\_bl: le\_tm\ M \neq bl\_tm\ M$ 
    and  $card\_ge: card\ (\Gamma\_tm\ M) \geq 4$ 
  obtains  $d\ e\ f\ b :: nat$ 
  where  $b \geq 1$ 
    and  $\forall w. set\ w \subseteq Sigma\_tm\ M$ 
       $\rightarrow accepts\_in\_time\_mttm\ M\ w\ (T\ (length\ w))$ 
       $\rightarrow accepts\_in\_time\_mttm$ 
         $(alphabet\_reduce\ M$ 
           $:: ('q \times 'a\ ar\_stage, sym4)\ mttm)$ 
           $(encode\_input\_ar\ (\Gamma\_tm\ M)\ (bl\_tm\ M)\ w)$ 
           $(d * b * T\ (length\ w) + e * b * length\ w + f))$ 
```

<proof>

Output alphabet: the reduced machine’s tape alphabet is the whole finite type *sym4*, of cardinality exactly four.

```
theorem alphabet_reduce_produces_alphabet_size_4:
  shows  $card\ (UNIV :: sym4\ set) = 4$ 
<proof>
```

Tape-count preservation: the reduction re-encodes the tape alphabet symbol by symbol without changing the number of tapes, so the reduced machine has exactly *M*’s tape count. This is the formal counterpart of the headline “tape count preserved”, and the point of contrast with a *tape-count* reduction such as Book–Greibach–Wegbreit.

```
theorem alphabet_reduce_preserves_tape_count:
  fixes  $M :: ('q, 'a)\ mttm$ 
  shows  $k\_tm\ (alphabet\_reduce\ M :: ('q \times 'a\ ar\_stage, sym4)\ mttm)$ 
     $= k\_tm\ M$ 
<proof>
```

end

```
theory AlphabetReduction_Reverse
  imports AlphabetReduction_Theorems
begin
```

3 Alphabet reduction: reverse language inclusion

The reverse leg of *alphabet_reduce_language_encode_input_ar* ($\Gamma_tm\ M$) ($bl_tm\ M$) $w \in Lang_mttm\ (alphabet_reduce\ M) \implies w \in Lang_mttm\ M$, the converse of the proven *alphabet_reduce_language_forward*.

Unlike AE's reverse arm, which restricts to *det_mttm* M and closes via chain uniqueness (Path C), AR proves this unconditionally — deterministic and nondeterministic M alike. The counterexample probe is negative: AR's nondeterminism gateway *ar_delta_compute* is a direct single-step embedding of *delta_tm* M (one substrate tuple per M -tuple, no AE-style *m_steps_buffered* slack), so every accepting M' -path decodes branch-by-branch to a genuine accepting M -path.

Strategy (Strategy B, direct backward inversion): a per-cycle backward step lemma reads the M -transition straight off the compute tuple on the given M' -path and reuses the forward arm's invariants *ar_simulates*, *ar_posk_consistent*, *ar_at_read_boundary* read backward; a backward chunked engine aggregates the per-cycle steps into an M -run. Throughout, b abbreviates *block_width* Γ (the per-symbol cell width).

3.1 Terminal accept configuration

Base case of the backward engine: the accept state *t_tm* (*alphabet_reduce* M) is terminal. A valid machine never steps from its accept state (substrate *mttm_step_src_neq_t*), and *alphabet_reduce* M is valid by *alphabet_reduce_wf*; its accept state is (*t_tm* M , *ar_accept_stage* (*bl_tm* M)) by *alphabet_reduce_accept*.

Itself currently uncalled: the backward engine kills reject boundaries mid-trace via *ar_reject_terminal*, not accept ones; retained as the documented half of the accept/reject terminal pair.

lemma *ar_accept_terminal*:
fixes $M :: ('q, 'a)\ mttm$
and $c\ c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM :: valid_mttm\ M$
and $card_ge :: card\ (\Gamma_tm\ M) \geq 4$
and $step :: (c, c') \in mttm_step\ (alphabet_reduce_delta\ M)$
shows $mt_state\ c \neq (t_tm\ M, ar_accept_stage\ (bl_tm\ M))$
<proof>

Reject companion of *ar_accept_terminal*: the reject state *r_tm* (*alphabet_reduce* M) = (*r_tm* M , *ar_reject_stage* (*bl_tm* M)) (by *alphabet_reduce_reject*) is terminal too (substrate *mttm_step_src_neq_r*). The backward engine uses it to kill a reject boundary reached mid-trace: the trace runs to the accept config, so a reject config can carry no outgoing step.

lemma *ar_reject_terminal*:
fixes $M :: ('q, 'a)\ mttm$

and $c \ c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{ valid_mttm } M$
and $\text{card_ge}: \text{card } (\Gamma_tm \ M) \geq 4$
and $\text{step}: (c, c') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
shows $\text{mt_state } c \neq (\text{r_tm } M, \text{ ar_reject_stage } (\text{bl_tm } M))$
 $\langle \text{proof} \rangle$

3.2 Source substep-tags partition the substep relations

Each of the five substep relations carries a uniform source substep-tag ($\text{fst } (\text{snd } s)$ for a source state $s = (q, \text{stg})$), and the five tags are pairwise distinct datatype constructors. This is the structural fact every step-inversion lemma rests on: a step whose source is at substep X can only come from the relation whose sources carry tag X .

lemma $\text{ar_delta_read_src}: (s, a, s', a', d) \in \text{ar_delta_read } M \implies \text{fst } (\text{snd } s) = \text{AR_SimRead}$
 $\langle \text{proof} \rangle$

lemma $\text{ar_delta_compute_src}: (s, a, s', a', d) \in \text{ar_delta_compute } M \implies \text{fst } (\text{snd } s) = \text{AR_SimCompute}$
 $\langle \text{proof} \rangle$

lemma $\text{ar_delta_write_src}: (s, a, s', a', d) \in \text{ar_delta_write } M \implies \text{fst } (\text{snd } s) = \text{AR_SimWrite}$
 $\langle \text{proof} \rangle$

lemma $\text{ar_delta_advance_src}: (s, a, s', a', d) \in \text{ar_delta_advance } M \implies \text{fst } (\text{snd } s) = \text{AR_SimAdvance}$
 $\langle \text{proof} \rangle$

lemma $\text{ar_delta_next_src}: (s, a, s', a', d) \in \text{ar_delta_next } M \implies \text{fst } (\text{snd } s) = \text{AR_SimNext}$
 $\langle \text{proof} \rangle$

3.3 Destination substep-tags carve the cycle's substep order

Each substep relation's destination tag is confined to a small set: read loops to itself or transitions to compute; compute is the unique non-deterministic step and lands at write; write loops or advances; advance loops or hands off to next; next closes the cycle (back to read) or dispatches to a halt-coerced configuration. These five facts encode the substep transition graph and underpin the chain-shape arguments used in the reverse arm's pinning lemmas.

lemma $\text{ar_delta_read_dest}: (s, a, s', a', d) \in \text{ar_delta_read } M \implies \text{fst } (\text{snd } s') = \text{AR_SimRead} \vee \text{fst } (\text{snd } s') = \text{AR_SimCompute}$
 $\langle \text{proof} \rangle$

lemma *ar_delta_compute_dest*:

$(s, a, s', a', d) \in \text{ar_delta_compute } M \implies \text{fst } (\text{snd } s') = \text{AR_SimWrite}$
 $\langle \text{proof} \rangle$

lemma *ar_delta_write_dest*:

$(s, a, s', a', d) \in \text{ar_delta_write } M \implies$
 $\text{fst } (\text{snd } s') = \text{AR_SimWrite} \vee \text{fst } (\text{snd } s') = \text{AR_SimAdvance}$
 $\langle \text{proof} \rangle$

lemma *ar_delta_advance_dest*:

$(s, a, s', a', d) \in \text{ar_delta_advance } M \implies$
 $\text{fst } (\text{snd } s') = \text{AR_SimAdvance} \vee \text{fst } (\text{snd } s') = \text{AR_SimNext}$
 $\langle \text{proof} \rangle$

lemma *ar_delta_next_dest*:

$(s, a, s', a', d) \in \text{ar_delta_next } M \implies$
 $\text{fst } (\text{snd } s') = \text{AR_SimRead} \vee \text{fst } (\text{snd } s') = \text{AR_HaltAccept}$
 $\vee \text{fst } (\text{snd } s') = \text{AR_HaltReject}$
 $\langle \text{proof} \rangle$

Union-disambiguation: a tuple in *alphabet_reduce_delta M* whose source is at *AR_SimCompute* must lie in the compute relation *ar_delta_compute M*. The intersection filters of *alphabet_reduce_delta* (δLE and the valid-stage guards) only shrink the union, so membership of the union is all we need; the other four source-tag lemmas rule out the other disjuncts by constructor-distinctness.

lemma *ar_delta_compute_from_src*:

assumes *mem*: $(s, a, s', a', d) \in \text{alphabet_reduce_delta } M$
and *src*: $\text{fst } (\text{snd } s) = \text{AR_SimCompute}$
shows $(s, a, s', a', d) \in \text{ar_delta_compute } M$
 $\langle \text{proof} \rangle$

3.4 Compute-step inversion

The keystone of the reverse arm: a single M' -step out of an *AR_SimCompute* configuration reads the simulated M -transition straight off the compute tuple. Because *ar_delta_compute* is a direct single-step embedding of *delta_tm M*, the inversion yields a genuine $(q, \text{buf}, q', m_{a'}, m_d) \in \text{delta_tm } M$ with no chain-uniqueness or determinism assumption — this is where AR's ND-generality is earned. Compute neither writes nor moves: the tape and head positions are unchanged (read symbol equals write symbol, direction N).

lemma *ar_compute_step_inv_sub*:

fixes $M :: ('q, 'a) \text{mttm}$
and $c' c'' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes *step*: $(c', c'') \in \text{mttm_step } (\text{ar_delta_compute } M)$

and $stg: mt_state\ c' = (q, AR_SimCompute, tk, i, buf, dvec, posk)$
obtains $q' m_a' m_d$ **where**
 $(q, buf, q', m_a', m_d) \in \delta_{tm}\ M$
and $mt_state\ c'' = (q', AR_SimWrite, 0, 0, m_a', m_d, posk)$
and $mt_tape\ c'' = mt_tape\ c'$
and $mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

The union-step face of the inversion: lift the union step into the compute sub-relation ($ar_step_compute_lift$) and invert there. Used by the forward walker preservation $ar_walker_step_from_at_compute$; the reverse cycle-close inverts the walker's own sub-relation compute step directly via $ar_compute_step_inv_sub$.

lemma $ar_compute_step_inv$:
fixes $M :: ('q, 'a)\ mttm$
and $c' c'' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $step: (c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
and $stg: mt_state\ c' = (q, AR_SimCompute, tk, i, buf, dvec, posk)$
obtains $q' m_a' m_d$ **where**
 $(q, buf, q', m_a', m_d) \in \delta_{tm}\ M$
and $mt_state\ c'' = (q', AR_SimWrite, 0, 0, m_a', m_d, posk)$
and $mt_tape\ c'' = mt_tape\ c'$
and $mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

3.5 Next-step inversion (cycle closure / halt dispatch)

Source disambiguation for $AR_SimNext$, mirroring
 $ar_delta_compute_from_src$.

lemma $ar_delta_next_from_src$:
assumes $mem: (s, a, s', a', d) \in alphabet_reduce_delta\ M$
and $src: fst\ (snd\ s) = AR_SimNext$
shows $(s, a, s', a', d) \in ar_delta_next\ M$
 $\langle proof \rangle$

A single M' -step out of an $AR_SimNext$ configuration neither writes nor moves, and dispatches on the simulated M -state q : continue to the next $AR_SimRead$ boundary when q is non-halting, or land in the accept/reject halt stage when q is M 's accept/reject state. The accept landing is exactly $t_tm\ (alphabet_reduce\ M)$.

lemma $ar_next_step_inv$:
fixes $M :: ('q, 'a)\ mttm$
and $c' c'' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $step: (c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
and $stg: mt_state\ c' = (q, AR_SimNext, tk, i, buf, dvec, posk)$
shows $mt_tape\ c'' = mt_tape\ c' \wedge mt_pos\ c'' = mt_pos\ c'$
 $\wedge ((q \notin \{t_tm\ M, r_tm\ M\})$
 $\wedge mt_state\ c'' = (q, AR_SimRead, 0, 0, buf, dvec, posk))$

$$\begin{aligned}
& \vee (q = t_tm\ M \\
& \quad \wedge mt_state\ c'' = (t_tm\ M, ar_accept_stage\ (bl_tm\ M))) \\
& \vee (q = r_tm\ M \\
& \quad \wedge mt_state\ c'' = (r_tm\ M, ar_reject_stage\ (bl_tm\ M)))
\end{aligned}$$

<proof>

3.6 Determinism of the read substep

The read relation is a *function* of (source state, read symbol): for a fixed source s and read vector a the target state, written vector, and direction are uniquely determined. The seven arms partition by the bit-counter i , then within an i -class by $posk\ tk / a\ tk / is_last_k\ M\ tk$. The one non-obvious exclusion is arm 4 ($i = Suc\ 0$) versus arms 6/7 ($i = Suc\ (b)$): these collide only if $b = 0$, ruled out by $block_width_pos$ (which is therefore load-bearing here, not decorative).

lemma $ar_delta_read_func$:
fixes $M :: ('q, 'a)\ mttm$
assumes $(s, a, s_1, a_1, d_1) \in ar_delta_read\ M$
and $(s, a, s_2, a_2, d_2) \in ar_delta_read\ M$
shows $(s_1, a_1, d_1) = (s_2, a_2, d_2)$
<proof>

Write is a function of (source, read vector): the LE arms ($buf\ tk = le$) split from the proper arms by the buf cell, and the proper back-walk / forward-write / boundary arms partition by the bit-counter ranges $[0, b-1]$, $[b, 2b-2]$, $\{2b-1\}$, separated arithmetically; is_last_k splits the last-tape arms.

lemma $ar_delta_write_func$:
fixes $M :: ('q, 'a)\ mttm$
assumes $(s, a, s_1, a_1, d_1) \in ar_delta_write\ M$
and $(s, a, s_2, a_2, d_2) \in ar_delta_write\ M$
shows $(s_1, a_1, d_1) = (s_2, a_2, d_2)$
<proof>

Advance is a function of (source, read vector). The stepping arm ($Suc\ i < ar_disp\ b\ (dvec\ tk)\ (posk\ tk)$) is excluded from the R -direction boundary sub-case by $ar_disp_dir.R_ = 0$ (the first ar_disp equation), and from the non- R boundary by $<$ versus $=$ on the displacement; is_last_k splits the two boundary arms.

lemma $ar_delta_advance_func$:
fixes $M :: ('q, 'a)\ mttm$
assumes $(s, a, s_1, a_1, d_1) \in ar_delta_advance\ M$
and $(s, a, s_2, a_2, d_2) \in ar_delta_advance\ M$
shows $(s_1, a_1, d_1) = (s_2, a_2, d_2)$
<proof>

Next is a function of the source: the continue / accept / reject arms

partition on the simulated M -state q by $q \notin \{t, r\} / q = t / q = r$, mutually exclusive precisely because $\text{valid_mttm } M$ supplies $t_tm M \neq r_tm M$.

Currently uncalled: the *next* substep is the cycle closer, handled by bespoke inversion (ar_next_step_inv) rather than functional pinning, so this member of the per-substep determinism family goes unused; kept to keep that family complete.

lemma $\text{ar_delta_next_func}$:
fixes $M :: ('q, 'a) \text{ mttm}$
assumes $vM: \text{valid_mttm } M$
and $(s, a, s_1, a_1, d_1) \in \text{ar_delta_next } M$
and $(s, a, s_2, a_2, d_2) \in \text{ar_delta_next } M$
shows $(s_1, a_1, d_1) = (s_2, a_2, d_2)$
 $\langle \text{proof} \rangle$

3.7 Source disambiguation for the remaining substeps

Source disambiguation for the remaining three substeps, completing the from_src family alongside $\text{ar_delta_compute_from_src} / \text{ar_delta_next_from_src}$.

lemma $\text{ar_delta_read_from_src}$:
assumes $(s, a, s', a', d) \in \text{alphabet_reduce_delta } M$
and $\text{fst } (\text{snd } s) = \text{AR_SimRead}$
shows $(s, a, s', a', d) \in \text{ar_delta_read } M$
 $\langle \text{proof} \rangle$

lemma $\text{ar_delta_write_from_src}$:
assumes $(s, a, s', a', d) \in \text{alphabet_reduce_delta } M$
and $\text{fst } (\text{snd } s) = \text{AR_SimWrite}$
shows $(s, a, s', a', d) \in \text{ar_delta_write } M$
 $\langle \text{proof} \rangle$

lemma $\text{ar_delta_advance_from_src}$:
assumes $(s, a, s', a', d) \in \text{alphabet_reduce_delta } M$
and $\text{fst } (\text{snd } s) = \text{AR_SimAdvance}$
shows $(s, a, s', a', d) \in \text{ar_delta_advance } M$
 $\langle \text{proof} \rangle$

Every $\text{alphabet_reduce_delta}$ tuple has its source at one of the five substep tags (the halt tags never appear as sources).

lemma $\text{alphabet_reduce_delta_src_tag}$:
assumes $(s, a, s', a', d) \in \text{alphabet_reduce_delta } M$
shows $\text{fst } (\text{snd } s) \in \{\text{AR_SimRead}, \text{AR_SimCompute}, \text{AR_SimWrite}, \text{AR_SimAdvance}, \text{AR_SimNext}\}$
 $\langle \text{proof} \rangle$

3.8 Substep step semantics — *mttm_step*-level lands-at

Lift the per-relation destination-tag lemmas (*ar_delta_X_dest*) up through *mttm_step*: an M' -step out of a configuration whose source idx is *AR_SimX* lands at a configuration whose idx is in X 's dest set. The five lemmas compose *mttm_step.cases* (extract the firing tuple), the *from_src* union-disambiguation helpers, and *ar_delta_X_dest*. Together they encode the cycle's substep transition graph at the level the substep-walker engine consumes: *SimRead-loop-or-compute*, *compute-to-write*, *write-loop-or-advance*, *advance-loop-or-next*, *next-to-read-or-halt*.

lemma *ar_step_from_SimRead_lands*:
assumes *step*: $(c, c') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
and *src*: $\text{fst } (\text{snd } (\text{mt_state } c)) = \text{AR_SimRead}$
shows $\text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimRead}$
 $\vee \text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimCompute}$
 $\langle \text{proof} \rangle$

The *compute* member of the five-lemma lands-at family above is currently uncalled: the compute substep is the nondeterministic branch point, handled by bespoke reconstruct-and-reuse reasoning rather than the generic lands-at lift. Retained to keep the substep transition graph complete.

lemma *ar_step_from_SimCompute_lands*:
assumes *step*: $(c, c') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
and *src*: $\text{fst } (\text{snd } (\text{mt_state } c)) = \text{AR_SimCompute}$
shows $\text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimWrite}$
 $\langle \text{proof} \rangle$

lemma *ar_step_from_SimWrite_lands*:
assumes *step*: $(c, c') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
and *src*: $\text{fst } (\text{snd } (\text{mt_state } c)) = \text{AR_SimWrite}$
shows $\text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimWrite}$
 $\vee \text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimAdvance}$
 $\langle \text{proof} \rangle$

lemma *ar_step_from_SimAdvance_lands*:
assumes *step*: $(c, c') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
and *src*: $\text{fst } (\text{snd } (\text{mt_state } c)) = \text{AR_SimAdvance}$
shows $\text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimAdvance}$
 $\vee \text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimNext}$
 $\langle \text{proof} \rangle$

lemma *ar_step_from_SimNext_lands*:
assumes *step*: $(c, c') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
and *src*: $\text{fst } (\text{snd } (\text{mt_state } c)) = \text{AR_SimNext}$
shows $\text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_SimRead}$
 $\vee \text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_HaltAccept}$
 $\vee \text{fst } (\text{snd } (\text{mt_state } c')) = \text{AR_HaltReject}$
 $\langle \text{proof} \rangle$

3.9 Walker invariants — per-substep stage predicates

The substep-walker is a per-step induction over the M' -path that tracks where in the cycle we are by reading the substep idx off each visited configuration's state. Six stage predicates carry the per-substep relationship between the cycle's source M -config cM and the current M' -configuration c' :

- $ar_walker_at_boundary\ M\ cM\ c'$: fresh cycle start — c' at $AR_SimRead$ boundary, the three forward invariants hold for cM .
- $ar_walker_in_read\ M\ cM\ c'$: mid-read-phase — reachable from a boundary by ar_delta_read -only steps, still at $AR_SimRead$.
- $ar_walker_at_compute\ M\ cM\ c'$: read complete — reachable from a boundary by ar_delta_read -only steps, now at $AR_SimCompute$. The next M' -step on the path extracts the M -tuple via $ar_compute_step_inv$.
- $ar_walker_in_write\ M\ cM\ c'$: M -tuple extracted, mid-write-phase — at $AR_SimWrite$.
- $ar_walker_in_advance\ M\ cM\ c'$: write done, mid-advance-phase — at $AR_SimAdvance$.
- $ar_walker_at_next\ M\ cM\ c'$: at $AR_SimNext$, about to dispatch to next boundary or halt via $ar_next_step_inv$.

The witness-chain formulation (rather than concrete per-state conditions) makes preservation lemmas mechanical: at a config with substep tag T , an M' -step fires the unique substep relation with src tag T (by $ar_delta_T_src + from_src$); extending the witness chain by one step preserves the invariant. Chain shape (no cycle-wrap before completing this cycle) follows from the witness chain living in the **specific** substep relation $mttm_step\ (ar_delta_T\ M)$, which by $ar_delta_T_src$ can only fire from sources at T — ruling out the wrap.

definition $ar_walker_at_boundary ::$

```

('q, 'a) mttm
  ⇒ ('a, 'q) mt_config
  ⇒ (sym4, 'q × 'a ar_stage) mt_config ⇒ bool where
   $ar\_walker\_at\_boundary\ M\ cM\ c' \longleftrightarrow$ 
     $ar\_simulates\ M\ cM\ c'$ 
   $\wedge\ ar\_posk\_consistent\ M\ cM\ c'$ 
   $\wedge\ ar\_at\_read\_boundary\ M\ c'$ 

```

definition $ar_walker_in_read ::$

```

('q, 'a) mttm

```

$$\begin{aligned}
&\Rightarrow ('a, 'q) \text{ mt_config} \\
&\Rightarrow (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config} \Rightarrow \text{bool} \textbf{ where} \\
&\text{ar_walker_in_read } M \text{ cM } c' \longleftrightarrow \\
&\quad \text{fst} (\text{snd} (\text{mt_state } c')) = \text{AR_SimRead} \\
&\wedge (\exists c_b \text{ m. ar_walker_at_boundary } M \text{ cM } c_b \\
&\quad \wedge (c_b, c') \in (\text{mttm_step} (\text{ar_delta_read } M)) \wedge m)
\end{aligned}$$

definition *ar_walker_at_compute* ::

$$\begin{aligned}
&('q, 'a) \text{ mttm} \\
&\Rightarrow ('a, 'q) \text{ mt_config} \\
&\Rightarrow (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config} \Rightarrow \text{bool} \textbf{ where} \\
&\text{ar_walker_at_compute } M \text{ cM } c' \longleftrightarrow \\
&\quad \text{fst} (\text{snd} (\text{mt_state } c')) = \text{AR_SimCompute} \\
&\wedge (\exists c_b \text{ m. ar_walker_at_boundary } M \text{ cM } c_b \\
&\quad \wedge (c_b, c') \in (\text{mttm_step} (\text{ar_delta_read } M)) \wedge m)
\end{aligned}$$

definition *ar_walker_in_write* ::

$$\begin{aligned}
&('q, 'a) \text{ mttm} \\
&\Rightarrow ('a, 'q) \text{ mt_config} \\
&\Rightarrow (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config} \Rightarrow \text{bool} \textbf{ where} \\
&\text{ar_walker_in_write } M \text{ cM } c' \longleftrightarrow \\
&\quad \text{fst} (\text{snd} (\text{mt_state } c')) = \text{AR_SimWrite} \\
&\wedge (\exists c_b \text{ c_w } m_r \text{ m_w.} \\
&\quad \text{ar_walker_at_boundary } M \text{ cM } c_b \\
&\quad \wedge (c_b, c_w) \in (\text{mttm_step} (\text{ar_delta_read } M)) \wedge m_r \\
&\quad \wedge \text{fst} (\text{snd} (\text{mt_state } c_w)) = \text{AR_SimCompute} \\
&\quad \wedge (\exists c_w_post. (c_w, c_w_post) \in \text{mttm_step} (\text{ar_delta_compute } M) \\
&\quad \wedge (c_w_post, c') \in (\text{mttm_step} (\text{ar_delta_write } M)) \wedge m_w))
\end{aligned}$$

definition *ar_walker_in_advance* ::

$$\begin{aligned}
&('q, 'a) \text{ mttm} \\
&\Rightarrow ('a, 'q) \text{ mt_config} \\
&\Rightarrow (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config} \Rightarrow \text{bool} \textbf{ where} \\
&\text{ar_walker_in_advance } M \text{ cM } c' \longleftrightarrow \\
&\quad \text{fst} (\text{snd} (\text{mt_state } c')) = \text{AR_SimAdvance} \\
&\wedge (\exists c_b \text{ c_w } c_a \text{ m_r } m_w \text{ m_a.} \\
&\quad \text{ar_walker_at_boundary } M \text{ cM } c_b \\
&\quad \wedge (c_b, c_w) \in (\text{mttm_step} (\text{ar_delta_read } M)) \wedge m_r \\
&\quad \wedge \text{fst} (\text{snd} (\text{mt_state } c_w)) = \text{AR_SimCompute} \\
&\quad \wedge \text{fst} (\text{snd} (\text{mt_state } c_a)) = \text{AR_SimAdvance} \\
&\quad \wedge (\exists c_w_post. (c_w, c_w_post) \in \text{mttm_step} (\text{ar_delta_compute } M) \\
&\quad \wedge (c_w_post, c_a) \in (\text{mttm_step} (\text{ar_delta_write } M)) \wedge m_w \\
&\quad \wedge (c_a, c') \in (\text{mttm_step} (\text{ar_delta_advance } M)) \wedge m_a))
\end{aligned}$$

definition *ar_walker_at_next* ::

$$\begin{aligned}
&('q, 'a) \text{ mttm} \\
&\Rightarrow ('a, 'q) \text{ mt_config} \\
&\Rightarrow (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config} \Rightarrow \text{bool} \textbf{ where}
\end{aligned}$$

$$\begin{aligned}
& ar_walker_at_next\ M\ cM\ c' \longleftrightarrow \\
& \quad fst\ (snd\ (mt_state\ c')) = AR_SimNext \\
& \quad \wedge\ (\exists\ c_b\ c_w\ c_a\ c_n\ m_r\ m_w\ m_a. \\
& \quad \quad ar_walker_at_boundary\ M\ cM\ c_b \\
& \quad \quad \wedge\ (c_b, c_w) \in (mttm_step\ (ar_delta_read\ M)) \wedge m_r \\
& \quad \quad \wedge\ fst\ (snd\ (mt_state\ c_w)) = AR_SimCompute \\
& \quad \quad \wedge\ fst\ (snd\ (mt_state\ c_a)) = AR_SimAdvance \\
& \quad \quad \wedge\ (\exists\ c_w_post. (c_w, c_w_post) \in mttm_step\ (ar_delta_compute\ M) \\
& \quad \quad \quad \wedge\ (c_w_post, c_a) \in (mttm_step\ (ar_delta_write\ M)) \wedge \\
m_w & \quad \quad \quad \wedge\ (c_a, c_n) \in (mttm_step\ (ar_delta_advance\ M)) \wedge m_a \\
& \quad \quad \quad \wedge\ c_n = c'))
\end{aligned}$$

3.10 Step lifts — *mttm_step* to specific substep

Five *mttm_step*-to-specific-substep lifts: an M' -step in *mttm_step* (*alphabet_reduce_delta* M) whose source carries substep tag T is in fact in the smaller *mttm_step* (*ar_delta_T* M). Each composes *mttm_step.cases* (destructure the step), the matching *ar_delta_T_from_src* helper (narrow the firing tuple by src-tag uniqueness), and *mttm_step.step* with *where* $ts = ts$ and $n = n$ instantiation (break the higher-order unification ambiguity inherent in *mttm_step.step*'s pattern when matched against concrete tuples). The walker preservation lemmas chain these lifts with the dest-tag dispatch (lands-at lemmas) to advance the witness chain by one step.

lemma *ar_step_read_lift*:
fixes $M :: ('q, 'a)\ mttm$
and $c'\ c'' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes *src*: $fst\ (snd\ (mt_state\ c')) = AR_SimRead$
and *step*: $(c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
shows $(c', c'') \in mttm_step\ (ar_delta_read\ M)$
<proof>

lemma *ar_step_compute_lift*:
fixes $M :: ('q, 'a)\ mttm$
and $c'\ c'' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes *src*: $fst\ (snd\ (mt_state\ c')) = AR_SimCompute$
and *step*: $(c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
shows $(c', c'') \in mttm_step\ (ar_delta_compute\ M)$
<proof>

lemma *ar_step_write_lift*:
fixes $M :: ('q, 'a)\ mttm$
and $c'\ c'' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes *src*: $fst\ (snd\ (mt_state\ c')) = AR_SimWrite$
and *step*: $(c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
shows $(c', c'') \in mttm_step\ (ar_delta_write\ M)$
<proof>

```

lemma ar_step_advance_lift:
  fixes  $M :: ('q, 'a) \text{mttm}$ 
    and  $c' c'' :: (\text{sym4}, 'q \times 'a \text{ ar\_stage}) \text{mt\_config}$ 
  assumes src:  $\text{fst} (\text{snd} (\text{mt\_state } c')) = \text{AR\_SimAdvance}$ 
    and step:  $(c', c'') \in \text{mttm\_step} (\text{alphabet\_reduce\_delta } M)$ 
  shows  $(c', c'') \in \text{mttm\_step} (\text{ar\_delta\_advance } M)$ 
<proof>

```

Existence-lift wrappers around the *ar_step_X_lift* lemmas for the three generic substeps (read, write, advance): convert an existential conclusion $\exists c''. (c', c'') \in \text{mttm_step} (\text{alphabet_reduce_delta } M) \wedge P c''$ into the same existential with the step in $\text{mttm_step} (\text{ar_delta_X } M)$, given the source tag of c' . These collapse the leaf-in-sub boilerplate (the *obtain ... using ar_step_X_lift[OF src ...] ... show ?thesis using ... by blast* scaffold) into a single application. The other two substeps carry no wrapper: compute is the nondeterministic branch and next closes the cycle, so both are handled by the cycle-close's reconstruct-and-reuse machinery rather than a generic lift.

```

lemma ar_exists_step_in_sub_read:
  fixes  $M :: ('q, 'a) \text{mttm}$ 
    and  $c' :: (\text{sym4}, 'q \times 'a \text{ ar\_stage}) \text{mt\_config}$ 
    and  $P :: (\text{sym4}, 'q \times 'a \text{ ar\_stage}) \text{mt\_config} \Rightarrow \text{bool}$ 
  assumes orig:  $\exists c''. (c', c'') \in \text{mttm\_step} (\text{alphabet\_reduce\_delta } M)$ 
     $\wedge P c''$ 
    and src:  $\text{fst} (\text{snd} (\text{mt\_state } c')) = \text{AR\_SimRead}$ 
  shows  $\exists c''. (c', c'') \in \text{mttm\_step} (\text{ar\_delta\_read } M) \wedge P c''$ 
<proof>

```

```

lemma ar_exists_step_in_sub_write:
  fixes  $M :: ('q, 'a) \text{mttm}$ 
    and  $c' :: (\text{sym4}, 'q \times 'a \text{ ar\_stage}) \text{mt\_config}$ 
    and  $P :: (\text{sym4}, 'q \times 'a \text{ ar\_stage}) \text{mt\_config} \Rightarrow \text{bool}$ 
  assumes orig:  $\exists c''. (c', c'') \in \text{mttm\_step} (\text{alphabet\_reduce\_delta } M)$ 
     $\wedge P c''$ 
    and src:  $\text{fst} (\text{snd} (\text{mt\_state } c')) = \text{AR\_SimWrite}$ 
  shows  $\exists c''. (c', c'') \in \text{mttm\_step} (\text{ar\_delta\_write } M) \wedge P c''$ 
<proof>

```

```

lemma ar_exists_step_in_sub_advance:
  fixes  $M :: ('q, 'a) \text{mttm}$ 
    and  $c' :: (\text{sym4}, 'q \times 'a \text{ ar\_stage}) \text{mt\_config}$ 
    and  $P :: (\text{sym4}, 'q \times 'a \text{ ar\_stage}) \text{mt\_config} \Rightarrow \text{bool}$ 
  assumes orig:  $\exists c''. (c', c'') \in \text{mttm\_step} (\text{alphabet\_reduce\_delta } M)$ 
     $\wedge P c''$ 
    and src:  $\text{fst} (\text{snd} (\text{mt\_state } c')) = \text{AR\_SimAdvance}$ 
  shows  $\exists c''. (c', c'') \in \text{mttm\_step} (\text{ar\_delta\_advance } M) \wedge P c''$ 
<proof>

```

3.11 Chain pinning in $mttm_step\ (ar_delta_read\ M)$

Two structural facts about chains in the read sub-relation: the sub-relation is **functional** (lifted from $ar_delta_read_func$ via $mttm_step$'s shape), so its m -step extension of any seed is unique; and it cannot fire from a source whose substep tag is $AR_SimCompute$ (by $ar_delta_read_src$). Together these pin a chain ending at $AR_SimCompute$ uniquely on both its length and its endpoint: if two chains in the sub-relation start at the same seed and both end at an $AR_SimCompute$ -tagged config, they coincide. This is what bridges the walker's by-construction R_read witness chain to the existence chain produced by the (re-mirrored) $ar_read_phase_in_sub$: the witness chain inherits the latter's stated endpoint state shape, including the load-bearing $buf = \lambda k. mt_tape\ cM\ k\ (mt_pos\ cM\ k)$.

lemma $mttm_step_ar_delta_read_func$:
fixes $M :: ('q, 'a)\ mttm$
and $c\ c_1\ c_2 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $h_1: (c, c_1) \in mttm_step\ (ar_delta_read\ M)$
and $h_2: (c, c_2) \in mttm_step\ (ar_delta_read\ M)$
shows $c_1 = c_2$
 $\langle proof \rangle$

lemma $chain_ar_delta_read_func$:
fixes $M :: ('q, 'a)\ mttm$
shows $(c, c_1) \in (mttm_step\ (ar_delta_read\ M))^{\wedge\wedge m}$
 $\implies (c, c_2) \in (mttm_step\ (ar_delta_read\ M))^{\wedge\wedge m}$
 $\implies c_1 = c_2$
 $\langle proof \rangle$

lemma $ar_delta_read_no_step_from_SimCompute$:
fixes $M :: ('q, 'a)\ mttm$
and $c\ c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $fst\ (snd\ (mt_state\ c)) = AR_SimCompute$
shows $(c, c') \notin mttm_step\ (ar_delta_read\ M)$
 $\langle proof \rangle$

lemma $chain_ar_delta_read_to_SimCompute_uniq$:
fixes $M :: ('q, 'a)\ mttm$
and $c\ c_1\ c_2 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $h_1: (c, c_1) \in (mttm_step\ (ar_delta_read\ M))^{\wedge\wedge m}$
and $h_2: (c, c_2) \in (mttm_step\ (ar_delta_read\ M))^{\wedge\wedge n}$
and $c_1cpu: fst\ (snd\ (mt_state\ c_1)) = AR_SimCompute$
and $c_2cpu: fst\ (snd\ (mt_state\ c_2)) = AR_SimCompute$
shows $m = n \wedge c_1 = c_2$
 $\langle proof \rangle$

Two parallel chain-pinning suites for the write and advance sub-relations, mirroring the read suite verbatim with ar_delta_write / $ar_delta_advance$ in place of ar_delta_read and $AR_SimAdvance$

/ *AR_SimNext* in place of *AR_SimCompute*. Functional projections (*ar_delta_write_func*, *ar_delta_advance_func*) and src uniqueness (*ar_delta_write_src*, *ar_delta_advance_src*) feed the same scaffold. These suites are used by the cycle-close bridging lemma to pin walker write/advance chains against the forward *ar_write_phase* / *ar_advance_phase* constructions.

```

lemma mttm_step_ar_delta_write_func:
  fixes M :: ('q, 'a) mttm
    and c c1 c2 :: (sym4, 'q × 'a ar_stage) mt_config
  assumes h1: (c, c1) ∈ mttm_step (ar_delta_write M)
    and h2: (c, c2) ∈ mttm_step (ar_delta_write M)
  shows c1 = c2
⟨proof⟩

```

```

lemma chain_ar_delta_write_func:
  fixes M :: ('q, 'a) mttm
  shows (c, c1) ∈ (mttm_step (ar_delta_write M)) ^m
    ⇒ (c, c2) ∈ (mttm_step (ar_delta_write M)) ^m
    ⇒ c1 = c2
⟨proof⟩

```

```

lemma ar_delta_write_no_step_from_SimAdvance:
  fixes M :: ('q, 'a) mttm
    and c c' :: (sym4, 'q × 'a ar_stage) mt_config
  assumes fst (snd (mt_state c)) = AR_SimAdvance
  shows (c, c') ∉ mttm_step (ar_delta_write M)
⟨proof⟩

```

```

lemma chain_ar_delta_write_to_SimAdvance_uniq:
  fixes M :: ('q, 'a) mttm
    and c c1 c2 :: (sym4, 'q × 'a ar_stage) mt_config
  assumes h1: (c, c1) ∈ (mttm_step (ar_delta_write M)) ^m
    and h2: (c, c2) ∈ (mttm_step (ar_delta_write M)) ^n
    and c1adv: fst (snd (mt_state c1)) = AR_SimAdvance
    and c2adv: fst (snd (mt_state c2)) = AR_SimAdvance
  shows m = n ∧ c1 = c2
⟨proof⟩

```

```

lemma mttm_step_ar_delta_advance_func:
  fixes M :: ('q, 'a) mttm
    and c c1 c2 :: (sym4, 'q × 'a ar_stage) mt_config
  assumes h1: (c, c1) ∈ mttm_step (ar_delta_advance M)
    and h2: (c, c2) ∈ mttm_step (ar_delta_advance M)
  shows c1 = c2
⟨proof⟩

```

```

lemma chain_ar_delta_advance_func:
  fixes M :: ('q, 'a) mttm

```

shows $(c, c_1) \in (\text{mttm_step } (\text{ar_delta_advance } M)) \hat{\wedge} m$
 $\implies (c, c_2) \in (\text{mttm_step } (\text{ar_delta_advance } M)) \hat{\wedge} m$
 $\implies c_1 = c_2$
 $\langle \text{proof} \rangle$

lemma *ar_delta_advance_no_step_from_SimNext*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c \ c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $\text{fst } (\text{snd } (\text{mt_state } c)) = \text{AR_SimNext}$
shows $(c, c') \notin \text{mttm_step } (\text{ar_delta_advance } M)$
 $\langle \text{proof} \rangle$

lemma *chain_ar_delta_advance_to_SimNext_uniq*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c \ c_1 \ c_2 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $h_1: (c, c_1) \in (\text{mttm_step } (\text{ar_delta_advance } M)) \hat{\wedge} m$
and $h_2: (c, c_2) \in (\text{mttm_step } (\text{ar_delta_advance } M)) \hat{\wedge} n$
and $c_1 \text{ next}: \text{fst } (\text{snd } (\text{mt_state } c_1)) = \text{AR_SimNext}$
and $c_2 \text{ next}: \text{fst } (\text{snd } (\text{mt_state } c_2)) = \text{AR_SimNext}$
shows $m = n \wedge c_1 = c_2$
 $\langle \text{proof} \rangle$

3.12 Read-phase leaves, sub-relation chain variants

For each single-step read-phase leaf (*ar_read_le_step*, *ar_read_le_finish_step*, *ar_read_lookback1_step*, *ar_read_lookback2_step*, *ar_read_bit_step*, *ar_read_bit_boundary_step*, *ar_read_bit_finish_step*), a companion lemma producing the step in *mttm_step (ar_delta_read M)* instead of *mttm_step (alphabet_reduce_delta M)*. Each variant uses the existing lemma to obtain the step, then lifts via *ar_step_read_lift* (the source tag is *AR_SimRead* by the leaf's *stg* hypothesis). No re-derivation of the step's effect — the existing leaf's stated post-state, post-tape, post-pos conclusions flow through verbatim.

lemma *ar_read_le_step_in_sub*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{notlast}: \neg \text{is_last_k } M \ tk$
and $\text{posk_le}: \text{posk } tk = \text{AR_AtLE}$
and $aLE: \text{mt_tape } c' \ tk \ (\text{mt_pos } c' \ tk) = LE4$
and $\text{vsrc}: \text{ar_valid_stage } (\Gamma_tm \ M) \ (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M)$
 $(\text{AR_SimRead}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$

shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_read } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, k_succ \text{ tk}, 0,$
 $\quad \text{buf}(\text{tk} := \text{le_tm } M), \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(\text{tk} := \text{Suc } (\text{mt_pos } c' \text{ tk}))$
 $\langle \text{proof} \rangle$

lemma *ar_read_le_finish_step_in_sub*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, \text{tk}, 0, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \text{ } M$
and $\text{last}: \text{is_last_k } M \text{ tk}$
and $\text{posk_le}: \text{posk } \text{tk} = \text{AR_AtLE}$
and $\text{aLE}: \text{mt_tape } c' \text{ tk } (\text{mt_pos } c' \text{ tk}) = \text{LE4}$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \text{ } M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, \text{tk}, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \text{ } M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \text{ } M)$
 $(\text{AR_SimRead}, \text{tk}, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_read } M)$
 $\wedge \text{mt_state } c'' = (\bar{q}, \text{AR_SimCompute}, k_unidx \text{ } 0, 0,$
 $\quad \text{buf}(\text{tk} := \text{le_tm } M), \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(\text{tk} := \text{Suc } (\text{mt_pos } c' \text{ tk}))$
 $\langle \text{proof} \rangle$

lemma *ar_read_lookback1_step_in_sub*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, \text{tk}, 0, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \text{ } M$
and $\text{posk_proper}: \text{posk } \text{tk} \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $\text{notLE}: \text{mt_tape } c' \text{ tk } (\text{mt_pos } c' \text{ tk}) \neq \text{LE4}$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \text{ } M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, \text{tk}, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \text{ } M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \text{ } M)$
 $(\text{AR_SimRead}, \text{tk}, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_read } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, \text{tk}, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(\text{tk} := \text{mt_pos } c' \text{ tk} - 1)$
 $\langle \text{proof} \rangle$

lemma *ar_read_lookback2_step_in_sub*:
fixes $M :: ('q, 'a) \text{mttm}$

and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, tk, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $\text{kge2}: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimRead}, tk, \text{Suc } 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_read } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, tk, \text{Suc } (\text{Suc } 0),$
 $\text{buf}(tk := \text{gamma_unenum } (\Gamma_tm \ M) (\text{bl_tm } M) 0), \text{dvec},$
 $\text{posk}(tk := \text{if } \text{mt_tape } c' tk (\text{mt_pos } c' tk) = \text{LE4}$
 $\text{then } \text{AR_AtFirstProper} \text{ else } \text{AR_AtFurtherProper}))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \text{Suc } (\text{mt_pos } c' tk))$
 $\langle \text{proof} \rangle$

lemma $\text{ar_read_bit_step_in_sub}$:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm \ M$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $\text{ilo}: 2 \leq i$
and $\text{ihi}: \text{Suc } i \leq \text{Suc } (\text{block_width } (\Gamma_tm \ M))$
and $\text{kge2}: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm \ M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimRead}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_read } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, tk, \text{Suc } i,$
 $\text{buf}(tk := \text{gamma_unenum } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(2 * \text{gamma_enum } (\Gamma_tm \ M) (\text{bl_tm } M) (\text{buf } tk)$
 $+ \text{bit_value } (\text{mt_tape } c' tk (\text{mt_pos } c' tk))))),$
 $\text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \text{Suc } (\text{mt_pos } c' tk))$
 $\langle \text{proof} \rangle$

lemma $\text{ar_read_bit_boundary_step_in_sub}$:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$

and $stg: mt_state\ c' = (q, AR_SimRead, tk, i, buf, dvec, posk)$
and $qQ: q \in Q_tm\ M$
and $notlast: \neg is_last_k\ M\ tk$
and $posk_proper: posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and $ieq: i = Suc\ (block_width\ (\Gamma_tm\ M))$
and $vsrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, i, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (ar_delta_read\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, k_succ\ tk, 0,$
 $buf(tk := gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(2 * gamma_enum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $+ bit_value\ (mt_tape\ c'\ tk\ (mt_pos\ c'\ tk))))),$
 $dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$
 $\langle proof \rangle$

lemma $ar_read_bit_finish_step_in_sub$:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, i, buf, dvec, posk)$
and $qQ: q \in Q_tm\ M$
and $last: is_last_k\ M\ tk$
and $posk_proper: posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and $ieq: i = Suc\ (block_width\ (\Gamma_tm\ M))$
and $vsrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, i, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (ar_delta_read\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $buf(tk := gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(2 * gamma_enum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $+ bit_value\ (mt_tape\ c'\ tk\ (mt_pos\ c'\ tk))))),$
 $dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := Suc\ (mt_pos\ c'\ tk))$
 $\langle proof \rangle$

3.13 Read-phase combinators, sub-relation chain variants

Multi-step combiner $_in_sub$ variants follow the same proof structure as the originals, but obtain their sub-chains from the leaf $_in_sub$ companions and compose via the generic $relpow_Suc_I2/relpow_add$ combinators

(which work over any relation, in particular $mttm_step\ (ar_delta_read\ M)$). All bookkeeping for tape, position, state shape transfers verbatim from the originals.

lemma $ar_read_bit_loop_in_sub$:

fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $posk_proper: posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, 2, buf0, dvec, posk)$
and $buf0_valid: \forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pos_base: mt_pos\ c'\ tk = base$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 2, buf0, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_read\ M))$
 $\wedge\wedge\ (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge\ mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm\ M)),$
 $buf0(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))\ (buf0\ tk)$
 $(map\ (\lambda m. mt_tape\ c'\ tk\ (base + m))$
 $[0..<block_width\ (\Gamma_tm\ M) - 1])),$
 $dvec, posk)$
 $\wedge\ mt_pos\ c''\ tk = base + (block_width\ (\Gamma_tm\ M) - 1)$
 $\wedge\ mt_tape\ c'' = mt_tape\ c'$
 $\wedge\ (\forall k'. k' \neq tk \longrightarrow mt_pos\ c''\ k' = mt_pos\ c'\ k')$
 $\langle proof \rangle$

lemma $ar_read_proper_prefix_in_sub$:

fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $posk_proper: posk\ tk \in \{AR_AtFirstProper, AR_AtFurtherProper\}$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $notLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$
and $base_pos: 0 < base$
and $pos_base: mt_pos\ c'\ tk = base$
and $vsr: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_read\ M))$
 $\wedge\wedge\ Suc\ (block_width\ (\Gamma_tm\ M))$
 $\wedge\ mt_state\ c'' = (q, AR_SimRead, tk, Suc\ (block_width\ (\Gamma_tm\ M)),$
 $buf(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))$
 $(gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ 0))$

$(\text{map } (\lambda m. \text{mt_tape } c' \text{ tk } (\text{base} + m))$
 $\quad [0..<\text{block_width } (\Gamma_tm \ M) - 1]),$
 $dvec,$
 $\text{posk}(\text{tk} := \text{if } \text{mt_tape } c' \text{ tk } (\text{base} - 1) = \text{LE4}$
 $\quad \text{then } \text{AR_AtFirstProper} \text{ else } \text{AR_AtFurtherProper}))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(\text{tk} := \text{base} + (\text{block_width } (\Gamma_tm \ M) -$
 $1))$
 $\langle \text{proof} \rangle$

lemma $\text{ar_read_proper_step_in_sub}$:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{notlast}: \neg \text{is_last_k } M \text{ tk}$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimRead}, \text{tk}, 0, \text{buf}, dvec, \text{posk})$
and $\text{notLE}: \text{mt_tape } c' \text{ tk } (\text{mt_pos } c' \text{ tk}) \neq \text{LE4}$
and $\text{base_pos}: 0 < \text{base}$
and $\text{pos_base}: \text{mt_pos } c' \text{ tk} = \text{base}$
and $\text{vsr}: \text{ar_valid_stage } (\Gamma_tm \ M) (\text{bl_tm } M)$
 $(\text{AR_SimRead}, \text{tk}, 0, \text{buf}, dvec, \text{posk})$
and $\text{pad0}: \forall j \geq k_tm \ M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src0}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimRead}, \text{tk}, 0, \text{buf}, dvec, \text{posk})$
shows $\exists c''. (c', c'') \in (\text{mttm_step } (\text{ar_delta_read } M))$
 $\quad \wedge \wedge (\text{block_width } (\Gamma_tm \ M) + 2)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimRead}, k_succ \text{ tk}, 0,$
 $\quad \text{buf}(\text{tk} := \text{foldl } (\text{ar_acc } (\Gamma_tm \ M) (\text{bl_tm } M))$
 $\quad (\text{gamma_unenum } (\Gamma_tm \ M) (\text{bl_tm } M) 0)$
 $\quad (\text{map } (\lambda m. \text{mt_tape } c' \text{ tk } (\text{base} + m))$
 $\quad [0..<\text{block_width } (\Gamma_tm \ M)])),$
 $dvec,$
 $\text{posk}(\text{tk} := \text{if } \text{mt_tape } c' \text{ tk } (\text{base} - 1) = \text{LE4}$
 $\quad \text{then } \text{AR_AtFirstProper} \text{ else } \text{AR_AtFurtherProper}))$
 $\wedge \text{mt_tape } c'' = \text{mt_tape } c'$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(\text{tk} := \text{base} + \text{block_width } (\Gamma_tm \ M))$
 $\langle \text{proof} \rangle$

lemma $\text{ar_read_proper_finish_step_in_sub}$:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{last}: \text{is_last_k } M \text{ tk}$
and $\text{posk_proper}: \text{posk } tk \in \{\text{AR_AtFirstProper}, \text{AR_AtFurtherProper}\}$

and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $notLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$
and $base_pos: 0 < base$
and $pos_base: mt_pos\ c'\ tk = base$
and $vsrvc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_read\ M))$
 $\wedge (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $buf(tk := foldl\ (ar_acc\ (\Gamma_tm\ M)\ (bl_tm\ M))$
 $(gamma_unenum\ (\Gamma_tm\ M)\ (bl_tm\ M)\ 0)$
 $(map\ (\lambda m. mt_tape\ c'\ tk\ (base + m))$
 $[0..<block_width\ (\Gamma_tm\ M)])),$
 $dvec,$
 $posk(tk := if\ mt_tape\ c'\ tk\ (base - 1) = LE4$
 $then\ AR_AtFirstProper\ else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + block_width\ (\Gamma_tm\ M))$
 $\langle proof \rangle$

lemma $ar_read_tape_step_in_sub$:

fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $tM :: nat \Rightarrow 'a$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $notlast: \neg is_last_k\ M\ tk$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $tcrr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $tM\ (mt_tape\ c'\ tk)$
and $ppos: mt_pos\ c'\ tk = sim_pos\ (block_width\ (\Gamma_tm\ M))\ p$
and $pkok: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $proper_mem: 1 \leq p \implies tM\ p \in \Gamma_tm\ M$
and $vsrvc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_read\ M))$
 $\wedge (if\ p = 0\ then\ 1\ else\ block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimRead, k_succ\ tk, 0,$
 $buf(tk := tM\ p), dvec,$
 $posk(tk := if\ p = 0\ then\ AR_AtLE$
 $else\ if\ p = 1\ then\ AR_AtFirstProper$
 $else\ AR_AtFurtherProper))$

$\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk :=$
 $\quad if\ p = 0\ then\ Suc\ 0$
 $\quad\quad else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
 $\langle proof \rangle$

lemma *ar_read_tape_finish_step_in_sub*:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $tM :: nat \Rightarrow 'a$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $last: is_last_k\ M\ tk$
and $stg: mt_state\ c' = (q, AR_SimRead, tk, 0, buf, dvec, posk)$
and $tcorr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $\quad tM\ (mt_tape\ c'\ tk)$
and $ppos: mt_pos\ c'\ tk = sim_pos\ (block_width\ (\Gamma_tm\ M))\ p$
and $pkok: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $proper_mem: 1 \leq p \implies tM\ p \in \Gamma_tm\ M$
and $vsrcc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $\quad (AR_SimRead, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $\quad (AR_SimRead, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_read\ M))$
 $\quad \wedge (if\ p = 0\ then\ 1\ else\ block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c'' = (q, AR_SimCompute, k_unidx\ 0, 0,$
 $\quad buf(tk := tM\ p), dvec,$
 $\quad posk(tk := if\ p = 0\ then\ AR_AtLE$
 $\quad\quad else\ if\ p = 1\ then\ AR_AtFirstProper$
 $\quad\quad else\ AR_AtFurtherProper))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk :=$
 $\quad if\ p = 0\ then\ Suc\ 0$
 $\quad\quad else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
 $\langle proof \rangle$

lemma *ar_read_prefix_in_sub*:
fixes $M :: ('q, 'a)\ mttm$
and $cM :: ('a, 'q)\ mt_config$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0: mt_state\ c0 = (q, AR_SimRead, 0, 0, buf0, dvec, posk0)$
and $tcorr: \forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)$

$(bl_tm\ M)$
 $(mt_tape\ cM\ k)\ (mt_tape\ c0\ k)$
and $ppos: \bigwedge k. mt_pos\ c0\ k = sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
and $pkok: \bigwedge k. posk0\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0$
and $tapeG: \bigwedge k. mt_tape\ cM\ k\ (mt_pos\ cM\ k) \in \Gamma_tm\ M$
and $bufG: \bigwedge k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, 0, 0, buf0, dvec, posk0)$
shows $j \leq k_tm\ M - 1 \implies$
 $(\exists c\ m. (c0, c) \in (mttm_step\ (ar_delta_read\ M)) \wedge m$
 $\wedge m \leq j * (block_width\ (\Gamma_tm\ M) + 2)$
 $\wedge mt_state\ c = (q, AR_SimRead, k_unidx\ j, 0,$
 $(\lambda k. \text{if } k_idx\ k < j \text{ then } mt_tape\ cM\ k\ (mt_pos\ cM\ k) \text{ else } buf0\ k),$
 $dvec,$
 $(\lambda k. \text{if } k_idx\ k < j$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } AR_AtLE$
 $\text{ else if } mt_pos\ cM\ k = 1 \text{ then } AR_AtFirstProper$
 $\text{ else } AR_AtFurtherProper)$
 $\text{ else } posk0\ k))$
 $\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. \text{if } k_idx\ k < j$
 $\text{ then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } Suc\ 0$
 $\text{ else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
 $\text{ + } block_width\ (\Gamma_tm\ M))$
 $\text{ else } mt_pos\ c0\ k))$
 $\langle proof \rangle$

lemma $ar_read_phase_in_sub:$

fixes $M :: ('q, 'a)\ mttm$
and $cM :: ('a, 'q)\ mt_config$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0: mt_state\ c0 = (q, AR_SimRead, 0, 0, buf0, dvec, posk0)$
and $tcrr: \forall k < k_tm\ M. ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)$
 $(bl_tm\ M)$
 $(mt_tape\ cM\ k)\ (mt_tape\ c0\ k)$
and $ppos: \bigwedge k. mt_pos\ c0\ k = sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k)$
and $pkok: \bigwedge k. posk0\ k = AR_AtLE \longleftrightarrow mt_pos\ cM\ k = 0$
and $tapeG: \bigwedge k. mt_tape\ cM\ k\ (mt_pos\ cM\ k) \in \Gamma_tm\ M$
and $bufG: \bigwedge k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimRead, 0, 0, buf0, dvec, posk0)$
shows $\exists c\ m. (c0, c) \in (mttm_step\ (ar_delta_read\ M)) \wedge m$

$$\begin{aligned}
& \wedge m \leq k_tm\ M * (block_width\ (\Gamma_tm\ M) + 2) \\
& \wedge mt_state\ c = (q, AR_SimCompute, k_unidx\ 0, 0, \\
& \quad (\lambda k. \text{if } k < k_tm\ M \text{ then } mt_tape\ cM\ k\ (mt_pos\ cM\ k) \text{ else } buf0\ k), \\
& \quad dvec, \\
& \quad (\lambda k. \text{if } k < k_tm\ M \\
& \quad \quad \text{then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } AR_AtLE \\
& \quad \quad \quad \text{else if } mt_pos\ cM\ k = 1 \text{ then } AR_AtFirstProper \\
& \quad \quad \quad \text{else } AR_AtFurtherProper) \\
& \quad \quad \text{else } posk0\ k)) \\
& \wedge mt_tape\ c = mt_tape\ c0 \\
& \wedge mt_pos\ c = (\lambda k. \text{if } k < k_tm\ M \\
& \quad \text{then } (\text{if } mt_pos\ cM\ k = 0 \text{ then } Suc\ 0 \\
& \quad \quad \text{else } sim_pos\ (block_width\ (\Gamma_tm\ M))\ (mt_pos\ cM\ k) + block_width \\
& \quad \quad (\Gamma_tm\ M)) \\
& \quad \text{else } mt_pos\ c0\ k) \\
& \langle proof \rangle
\end{aligned}$$

3.14 Write-phase leaves, sub-relation chain variants

Six write-phase leaves mirror to the sub-relation $mttm_step\ (ar_delta_write\ M)$ via $ar_step_write_lift$, parallel to the seven read leaves at the earlier subsection. Each variant obtains the step from the original leaf, derives the $AR_SimWrite$ source tag from the stg hypothesis, and lifts via $ar_step_write_lift$.

lemma $ar_write_le_step_in_sub$:
fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage) mt_config$
assumes vM : $valid_mttm\ M$
and stg : $mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and qQ : $q \in Q_tm\ M$
and $poskLE$: $posk\ tk = AR_AtLE$
and $notlast$: $\neg is_last_k\ M\ tk$
and $usrc$: $ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
and pad_blank : $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded$: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (ar_delta_write\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimWrite, k_succ\ tk, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

lemma $ar_write_le_finish_step_in_sub$:
fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage) mt_config$
assumes vM : $valid_mttm\ M$
and stg : $mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$

and $qQ: q \in Q_tm\ M$
and $poskLE: posk\ tk = AR_AtLE$
and $last: is_last_k\ M\ tk$
and $vsrsc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (ar_delta_write\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

lemma $ar_write_walk_step_in_sub$:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $stg: mt_state\ c' = (q, AR_SimWrite, tk, i, buf, dvec, posk)$
and $qQ: q \in Q_tm\ M$
and $poskproper: posk\ tk \neq AR_AtLE$
and $notLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$
and $step_le: Suc\ i \leq block_width\ (\Gamma_tm\ M)$
and $vsrsc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (ar_delta_write\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimWrite, tk, Suc\ i, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := mt_pos\ c'\ tk - 1)$
 $\langle proof \rangle$

lemma $ar_write_bit_step_in_sub$:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $stg: mt_state\ c' = (q, AR_SimWrite, tk, i, buf, dvec, posk)$
and $qQ: q \in Q_tm\ M$
and $poskproper: posk\ tk \neq AR_AtLE$
and $notLE: mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$
and $ilo: block_width\ (\Gamma_tm\ M) \leq i$
and $ihi: Suc\ i < 2 * block_width\ (\Gamma_tm\ M)$
and $vsrsc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk)$
and $pad_blank: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src_bounded: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, i, buf, dvec, posk)$

shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_write } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimWrite}, tk, \text{Suc } i, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = (\text{mt_tape } c')(tk :=$
 $\quad (\text{mt_tape } c' tk)(\text{mt_pos } c' tk :=$
 $\quad \quad \text{write_bit } (\Gamma_tm M) (\text{bl_tm } M) (\text{buf } tk)$
 $\quad \quad (i - \text{block_width } (\Gamma_tm M))))$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \text{Suc } (\text{mt_pos } c' tk))$
 <proof>

lemma *ar_write_bit_boundary_step_in_sub*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm M$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{notLE}: \text{mt_tape } c' tk (\text{mt_pos } c' tk) \neq \text{LE4}$
and $\text{notlast}: \neg \text{is_last_k } M tk$
and $\text{ihi}: \text{Suc } i = 2 * \text{block_width } (\Gamma_tm M)$
and $\text{vsrce}: \text{ar_valid_stage } (\Gamma_tm M) (\text{bl_tm } M)$
 $\quad (\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm M)$
 $\quad (\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_write } M)$
 $\wedge \text{mt_state } c'' = (q, \text{AR_SimWrite}, k_succ tk, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c'' = (\text{mt_tape } c')(tk :=$
 $\quad (\text{mt_tape } c' tk)(\text{mt_pos } c' tk :=$
 $\quad \quad \text{write_bit } (\Gamma_tm M) (\text{bl_tm } M) (\text{buf } tk)$
 $\quad \quad (i - \text{block_width } (\Gamma_tm M))))$
 $\wedge \text{mt_pos } c'' = (\text{mt_pos } c')(tk := \text{Suc } (\text{mt_pos } c' tk))$
 <proof>

lemma *ar_write_bit_finish_step_in_sub*:
fixes $M :: ('q, 'a) \text{mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ar_stage}) \text{mt_config}$
assumes $vM: \text{valid_mttm } M$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $qQ: q \in Q_tm M$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{notLE}: \text{mt_tape } c' tk (\text{mt_pos } c' tk) \neq \text{LE4}$
and $\text{last}: \text{is_last_k } M tk$
and $\text{ihi}: \text{Suc } i = 2 * \text{block_width } (\Gamma_tm M)$
and $\text{vsrce}: \text{ar_valid_stage } (\Gamma_tm M) (\text{bl_tm } M)$
 $\quad (\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
and $\text{pad_blank}: \forall j \geq k_tm M. \text{mt_tape } c' j (\text{mt_pos } c' j) = \text{BLANK4}$
and $\text{src_bounded}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm M)$
 $\quad (\text{AR_SimWrite}, tk, i, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c''. (c', c'') \in \text{mttm_step } (\text{ar_delta_write } M)$

$$\begin{aligned}
& \wedge \text{mt_state } c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk) \\
& \wedge \text{mt_tape } c'' = (\text{mt_tape } c') (tk := \\
& \quad (\text{mt_tape } c' tk) (\text{mt_pos } c' tk := \\
& \quad \quad \text{write_bit } (\Gamma_tm\ M) (bl_tm\ M) (buf\ tk) \\
& \quad \quad (i - \text{block_width } (\Gamma_tm\ M)))) \\
& \wedge \text{mt_pos } c'' = (\text{mt_pos } c') (tk := \text{Suc } (\text{mt_pos } c' tk))
\end{aligned}$$

$\langle \text{proof} \rangle$

3.15 Advance-phase leaves, sub-relation chain variants

Three advance-phase leaves ($ar_advance_walk_step$, $ar_advance_boundary_step$, $ar_advance_finish_step$) mirror to the sub-relation $mttm_step$ ($ar_delta_advance\ M$) via $ar_exists_step_in_sub_advance$, parallel to the write and read leaves above. Each variant derives the $AR_SimAdvance$ source tag from stg , then composes the original leaf with the existence lifter.

lemma $ar_advance_walk_step_in_sub$:

fixes $M :: ('q, 'a) mttm$

and $c' :: (sym4, 'q \times 'a\ ar_stage) mt_config$

assumes $vM: \text{valid_mttm } M$

and $stg: \text{mt_state } c' = (q, AR_SimAdvance, tk, i, buf, dvec, posk)$

and $qQ: q \in Q_tm\ M$

and $notLE: \text{mt_tape } c' tk (\text{mt_pos } c' tk) \neq LE4$

and $step_lt: \text{Suc } i < ar_disp (\text{block_width } (\Gamma_tm\ M)) (dvec\ tk) (posk\ tk)$

and $vsrc: ar_valid_stage (\Gamma_tm\ M) (bl_tm\ M)$
 $(AR_SimAdvance, tk, i, buf, dvec, posk)$

and $pad_blank: \forall j \geq k_tm\ M. \text{mt_tape } c' j (\text{mt_pos } c' j) = BLANK4$

and $src_bounded: ar_stage_bounded (bl_tm\ M) (k_tm\ M)$
 $(AR_SimAdvance, tk, i, buf, dvec, posk)$

shows $\exists c''. (c', c'') \in mttm_step (ar_delta_advance\ M)$

$\wedge \text{mt_state } c'' = (q, AR_SimAdvance, tk, \text{Suc } i, buf, dvec, posk)$

$\wedge \text{mt_tape } c'' = \text{mt_tape } c'$

$\wedge \text{mt_pos } c'' = (\text{mt_pos } c') (tk := \text{mt_pos } c' tk - 1)$

$\langle \text{proof} \rangle$

lemma $ar_advance_boundary_step_in_sub$:

fixes $M :: ('q, 'a) mttm$

and $c' :: (sym4, 'q \times 'a\ ar_stage) mt_config$

assumes $vM: \text{valid_mttm } M$

and $stg: \text{mt_state } c' = (q, AR_SimAdvance, tk, i, buf, dvec, posk)$

and $qQ: q \in Q_tm\ M$

and $notlast: \neg is_last_k\ M\ tk$

and $fire: (dvec\ tk = dir.R \wedge i = 0)$
 $\vee (dvec\ tk \neq dir.R$
 $\wedge \text{mt_tape } c' tk (\text{mt_pos } c' tk) \neq LE4$
 $\wedge \text{Suc } i = ar_disp (\text{block_width } (\Gamma_tm\ M)) (dvec\ tk) (posk\ tk))$

and $vsrc: ar_valid_stage (\Gamma_tm\ M) (bl_tm\ M)$
 $(AR_SimAdvance, tk, i, buf, dvec, posk)$

and *pad_blank*: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and *src_bounded*: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (ar_delta_advance\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_succ\ tk, 0, buf, dvec,$
 $posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk)))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c'$
 $else\ (mt_pos\ c')(tk := mt_pos\ c'\ tk - 1))$
 $\langle proof \rangle$

lemma *ar_advance_finish_step_in_sub*:
fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $stg: mt_state\ c' = (q, AR_SimAdvance, tk, i, buf, dvec, posk)$
and $qQ: q \in Q_tm\ M$
and $last: is_last_k\ M\ tk$
and $fire: (dvec\ tk = dir.R \wedge i = 0)$
 $\vee (dvec\ tk \neq dir.R$
 $\wedge mt_tape\ c'\ tk\ (mt_pos\ c'\ tk) \neq LE4$
 $\wedge Suc\ i = ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk))$
and $usrc: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimAdvance, tk, i, buf, dvec, posk)$
and *pad_blank*: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and *src_bounded*: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, i, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in mttm_step\ (ar_delta_advance\ M)$
 $\wedge mt_state\ c'' = (q, AR_SimNext, k_unidx\ 0, 0, buf, dvec,$
 $posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk)))$
 $\wedge mt_tape\ c'' = mt_tape\ c'$
 $\wedge mt_pos\ c'' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c'$
 $else\ (mt_pos\ c')(tk := mt_pos\ c'\ tk - 1))$
 $\langle proof \rangle$

3.16 Write-phase combinators, sub-relation chain variants

Write-phase multi-step combiner *_in_sub* variants follow the same proof structure as the originals, obtaining sub-chains from the write-leaf *_in_sub* companions and composing via the generic *relpow_Suc_I2* / *relpow_invariant_chain* combinators (which work over any relation, in particular *mttm_step* (*ar_delta_write* *M*)).

lemma *ar_write_back_loop_in_sub*:
fixes $M :: ('q, 'a)\ mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$

and *poskproper*: *posk tk* $\neq$ *AR_AtLE*
and *stg*: *mt_state c0* = (*q*, *AR_SimWrite*, *tk*, 0, *buf*, *dvec*, *posk*)
and *buf_valid*: $\forall k. \text{buf } k \in \Gamma_tm\ M \cup \{\text{bl_tm } M\}$
and *pos_base*: *mt_pos c0 tk* = *base* + *block_width* ($\Gamma_tm\ M$)
and *notLE*: $\bigwedge m. \llbracket 1 \leq m; m \leq \text{block_width } (\Gamma_tm\ M) \rrbracket$
 $\implies \text{mt_tape } c0\ tk\ (\text{base} + m) \neq \text{LE4}$
and *pad0*: $\forall j \geq k_tm\ M. \text{mt_tape } c0\ j\ (\text{mt_pos } c0\ j) = \text{BLANK4}$
and *src0*: *ar_stage_bounded* (*bl_tm M*) (*k_tm M*)
(*AR_SimWrite*, *tk*, 0, *buf*, *dvec*, *posk*)
shows $\exists c'. (c0, c') \in (\text{mttm_step } (\text{ar_delta_write } M))$
 $\wedge \wedge \text{block_width } (\Gamma_tm\ M)$
 $\wedge \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, \text{block_width } (\Gamma_tm\ M),$
 $\text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c'\ tk = \text{base}$
 $\wedge \text{mt_tape } c' = \text{mt_tape } c0$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c'\ k' = \text{mt_pos } c0\ k')$
<proof>

lemma *ar_write_fwd_loop_in_sub*:
fixes *M* :: ('q, 'a) *mttm*
and *c0* :: (*sym4*, 'q $\times$ 'a *ar_stage*) *mt_config*
assumes *vM*: *valid_mttm M*
and *qQ*: *q* $\in Q_tm\ M$
and *kge2*: $2 \leq \text{block_width } (\Gamma_tm\ M)$
and *poskproper*: *posk tk* $\neq$ *AR_AtLE*
and *stg*: *mt_state c0* = (*q*, *AR_SimWrite*, *tk*, *block_width* ($\Gamma_tm\ M$),
buf, *dvec*, *posk*)
and *buf_valid*: $\forall k. \text{buf } k \in \Gamma_tm\ M \cup \{\text{bl_tm } M\}$
and *pos_base*: *mt_pos c0 tk* = *base*
and *notLE*: $\bigwedge m. m < \text{block_width } (\Gamma_tm\ M)$
 $\implies \text{mt_tape } c0\ tk\ (\text{base} + m) \neq \text{LE4}$
and *pad0*: $\forall j \geq k_tm\ M. \text{mt_tape } c0\ j\ (\text{mt_pos } c0\ j) = \text{BLANK4}$
and *src0*: *ar_stage_bounded* (*bl_tm M*) (*k_tm M*)
(*AR_SimWrite*, *tk*, *block_width* ($\Gamma_tm\ M$), *buf*, *dvec*, *posk*)
shows $\exists c'. (c0, c') \in (\text{mttm_step } (\text{ar_delta_write } M))$
 $\wedge \wedge (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\wedge \text{mt_state } c' = (q, \text{AR_SimWrite}, tk,$
 $\text{block_width } (\Gamma_tm\ M) + (\text{block_width } (\Gamma_tm\ M) - 1),$
 $\text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_pos } c'\ tk = \text{base} + (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\wedge \text{mt_tape } c'\ tk = (\lambda \text{pos.}$
 $\text{if } \text{base} \leq \text{pos} \wedge \text{pos} < \text{base} + (\text{block_width } (\Gamma_tm\ M) - 1)$
 $\text{then } \text{write_bit } (\Gamma_tm\ M)\ (\text{bl_tm } M)\ (\text{buf } tk)\ (\text{pos} - \text{base})$
 $\text{else } \text{mt_tape } c0\ tk\ \text{pos})$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_tape } c'\ k' = \text{mt_tape } c0\ k')$
 $\wedge (\forall k'. k' \neq tk \longrightarrow \text{mt_pos } c'\ k' = \text{mt_pos } c0\ k')$
<proof>

The proper-cell single-tape write prefix, re-mirrored into *mttm_step (ar_delta_write M)*. Composes the back-walk loop

(*ar_write_back_loop_in_sub*) and the forward bit-write loop (*ar_write_fwd_loop_in_sub*) by *relcompI* + *relpow_add*; since both *_in_sub* sub-combiners carry field-for-field the same output contract as the originals, the composition and the two *ext* reassemblies (tape, pos) transfer verbatim with only the relation and the two helper calls changed.

lemma *ar_write_proper_prefix_in_sub*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{poskproper}: \text{posk } tk \neq \text{AR_AtLE}$
and $\text{stg}: \text{mt_state } c' = (q, \text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\}$
and $\text{pos_base}: \text{mt_pos } c' \ tk = \text{base} + \text{block_width } (\Gamma_tm \ M)$
and $\text{notLE}: \bigwedge m. m \leq \text{block_width } (\Gamma_tm \ M) \implies \text{mt_tape } c' \ tk \ (\text{base} + m) \neq \text{LE4}$
and $\text{pad0}: \forall j \geq k_tm \ M. \text{mt_tape } c' \ j \ (\text{mt_pos } c' \ j) = \text{BLANK4}$
and $\text{src0}: \text{ar_stage_bounded } (\text{bl_tm } M) \ (k_tm \ M) \ (\text{AR_SimWrite}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c2. (c', c2) \in (\text{mttm_step } (\text{ar_delta_write } M))$
 $\quad \quad \quad \wedge \wedge (\text{block_width } (\Gamma_tm \ M) + (\text{block_width } (\Gamma_tm \ M) - 1))$
 $\quad \quad \quad \wedge \text{mt_state } c2 = (q, \text{AR_SimWrite}, tk,$
 $\quad \quad \quad \quad \text{block_width } (\Gamma_tm \ M) + (\text{block_width } (\Gamma_tm \ M) - 1),$
 $\quad \quad \quad \quad \text{buf}, \text{dvec}, \text{posk})$
 $\quad \quad \quad \wedge \text{mt_tape } c2 = (\text{mt_tape } c') (tk := (\lambda \text{pos.}$
 $\quad \quad \quad \quad \text{if } \text{base} \leq \text{pos} \wedge \text{pos} < \text{base} + (\text{block_width } (\Gamma_tm \ M) - 1)$
 $\quad \quad \quad \quad \text{then } \text{write_bit } (\Gamma_tm \ M) \ (\text{bl_tm } M) \ (\text{buf } tk) \ (\text{pos} - \text{base})$
 $\quad \quad \quad \quad \text{else } \text{mt_tape } c' \ tk \ \text{pos}))$
 $\quad \quad \quad \wedge \text{mt_pos } c2 = (\text{mt_pos } c') (tk := \text{base} + (\text{block_width } (\Gamma_tm \ M) - 1))$
 $\quad \quad \quad \langle \text{proof} \rangle$

The proper-cell single-tape write, non-last tape, re-mirrored into *mttm_step* (*ar_delta_write* M). As *ar_write_proper_prefix_in_sub* followed by one bit-boundary step (*ar_write_bit_boundary_step_in_sub*), composing by *relpow_Suc_I*. The shared *write_block_extend* helper is relation-agnostic (pure *fun_upd* arithmetic) and reused verbatim; the body transfers from the original with only the relation and the two helper references changed.

lemma *ar_write_proper_step_in_sub*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $c' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $\text{notlast}: \neg \text{is_last_k } M \ tk$

and *poskproper*: $\text{posk } tk \neq AR_AtLE$
and *stg*: $mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and *buf_valid*: $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and *pos_base*: $mt_pos\ c'\ tk = base + block_width\ (\Gamma_tm\ M)$
and *notLE*: $\bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$
 $\implies mt_tape\ c'\ tk\ (base + m) \neq LE4$
and *pad0*: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and *src0*: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_write\ M))$
 $\wedge (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimWrite, k_succ\ tk, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = (mt_tape\ c')(tk := (\lambda pos.$
 $\quad if\ base \leq pos \wedge pos < base + block_width\ (\Gamma_tm\ M)$
 $\quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos - base)$
 $\quad else\ mt_tape\ c'\ tk\ pos))$
 $\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + block_width\ (\Gamma_tm\ M))$
<proof>

The proper-cell single-tape write, last tape, re-mirrored into *mttm_step* (*ar_delta_write* *M*). As *ar_write_proper_step_in_sub* but *tk* is the last tape, so the closing step is *ar_write_bit_finish_step_in_sub*: after the last-cell write the phase transitions to *AR_SimAdvance* with the current-tape field reset to *k_unidx 0*. Same *2b*-step block write and head return to *base + b*; body transfers verbatim with only the relation and the two helper references changed.

lemma *ar_write_proper_finish_step_in_sub*:
fixes *M* :: ('q, 'a) *mttm*
and *c'* :: (*sym4*, 'q $\times$ 'a *ar_stage*) *mt_config*
assumes *vM*: *valid_mttm* *M*
and *qQ*: $q \in Q_tm\ M$
and *kge2*: $2 \leq block_width\ (\Gamma_tm\ M)$
and *last*: *is_last_k* *M* *tk*
and *poskproper*: $\text{posk } tk \neq AR_AtLE$
and *stg*: $mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and *buf_valid*: $\forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and *pos_base*: $mt_pos\ c'\ tk = base + block_width\ (\Gamma_tm\ M)$
and *notLE*: $\bigwedge m. m \leq block_width\ (\Gamma_tm\ M)$
 $\implies mt_tape\ c'\ tk\ (base + m) \neq LE4$
and *pad0*: $\forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and *src0*: $ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_write\ M))$
 $\wedge (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = (mt_tape\ c')(tk := (\lambda pos.$
 $\quad if\ base \leq pos \wedge pos < base + block_width\ (\Gamma_tm\ M)$
 $\quad then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)\ (pos - base)$
 $\quad else\ mt_tape\ c'\ tk\ pos))$

$\wedge mt_pos\ c'' = (mt_pos\ c')(tk := base + block_width\ (\Gamma_tm\ M))$
 $\langle proof \rangle$

The unified single-tape write, non-last tape, re-mirrored into $mttm_step$ ($ar_delta_write\ M$): the LE/proper dispatch on $posk\ tk = AR_AtLE$ (pinned by $poskle$ to $p = 0$). The LE arm ($ar_write_le_step_in_sub$, 1 step) hands off untouched; the proper arm ($ar_write_proper_step_in_sub$, $2b$ steps) overwrites the b -cell block. Head invariant in both arms. The $\neq LE4$ facts come from the input correspondence $tcorr$; that derivation is relation-agnostic and transfers verbatim along with the relation and two helper references changing.

lemma $ar_write_tape_step_in_sub$:

fixes $M :: ('q, 'a)\ mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $tM :: nat \Rightarrow 'a$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $notlast: \neg is_last_k\ M\ tk$
and $stg: mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and $tcorr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $tM\ (mt_tape\ c'\ tk)$
and $ppos: mt_pos\ c'\ tk = (if\ p = 0\ then\ Suc\ 0$
 $else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
and $poskle: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $vsr: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_write\ M))$
 $\wedge\wedge (if\ p = 0\ then\ 1\ else\ 2 * block_width\ (\Gamma_tm\ M))$
 $\wedge\ mt_state\ c'' = (q, AR_SimWrite, k_succ\ tk, 0, buf, dvec, posk)$
 $\wedge\ mt_tape\ c'' = (if\ p = 0\ then\ mt_tape\ c'$
 $else\ (mt_tape\ c')(tk := (\lambda pos.$
 $if\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p \leq pos$
 $\wedge\ pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width$
 $(\Gamma_tm\ M)$
 $then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $(pos - sim_pos\ (block_width\ (\Gamma_tm\ M))\ p)$
 $else\ mt_tape\ c'\ tk\ pos)))$
 $\wedge\ mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

The unified single-tape write, last tape, re-mirrored into $mttm_step$ ($ar_delta_write\ M$): as $ar_write_tape_step_in_sub$ but tk is the last tape, so both arms transition to $AR_SimAdvance$ (current-tape field reset to

$k_unidx\ 0$): the LE arm via $ar_write_le_finish_step_in_sub$, the proper arm via $ar_write_proper_finish_step_in_sub$. Same tape edit and head-invariance; body transfers verbatim with only the relation and the two helper references changed.

lemma $ar_write_tape_finish_step_in_sub$:
fixes $M :: ('q, 'a) mttm$
and $c' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
and $tM :: nat \Rightarrow 'a$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $last: is_last_k\ M\ tk$
and $stg: mt_state\ c' = (q, AR_SimWrite, tk, 0, buf, dvec, posk)$
and $tcorr: ar_tape_correspondence\ (\Gamma_tm\ M)\ (le_tm\ M)\ (bl_tm\ M)$
 $tM\ (mt_tape\ c'\ tk)$
and $ppos: mt_pos\ c'\ tk = (if\ p = 0\ then\ Suc\ 0$
 $else\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width\ (\Gamma_tm$
 $M))$
and $poskle: posk\ tk = AR_AtLE \longleftrightarrow p = 0$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $vsrce: ar_valid_stage\ (\Gamma_tm\ M)\ (bl_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c'\ j\ (mt_pos\ c'\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimWrite, tk, 0, buf, dvec, posk)$
shows $\exists c''. (c', c'') \in (mttm_step\ (ar_delta_write\ M))$
 $\wedge (if\ p = 0\ then\ 1\ else\ 2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c'' = (q, AR_SimAdvance, k_unidx\ 0, 0, buf, dvec, posk)$
 $\wedge mt_tape\ c'' = (if\ p = 0\ then\ mt_tape\ c'$
 $else\ (mt_tape\ c')(tk := (\lambda pos.$
 $if\ sim_pos\ (block_width\ (\Gamma_tm\ M))\ p \leq pos$
 $\wedge pos < sim_pos\ (block_width\ (\Gamma_tm\ M))\ p + block_width$
 $(\Gamma_tm\ M)$
 $then\ write_bit\ (\Gamma_tm\ M)\ (bl_tm\ M)\ (buf\ tk)$
 $(pos - sim_pos\ (block_width\ (\Gamma_tm\ M))\ p)$
 $else\ mt_tape\ c'\ tk\ pos)))$
 $\wedge mt_pos\ c'' = mt_pos\ c'$
 $\langle proof \rangle$

The write-phase prefix walk, re-mirrored into $mttm_step\ (ar_delta_write\ M)$: from the write boundary, iterate the unified non-last per-tape write $ar_write_tape_step_in_sub$ over the first j tapes (all non-last), landing back at $AR_SimWrite$ on tape $k_unidx\ j$. The induction on j , the split tape descriptor, and the per-tape $tcorr$ /position bookkeeping transfer verbatim from the original; only the relation and the one helper reference change. The internal IH is already over $ar_delta_write\ M$.

lemma $ar_write_prefix_in_sub$:
fixes $M :: ('q, 'a) mttm$

and $cM :: ('a, 'q) \text{ mt_config}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $stg0: \text{mt_state } c0 = (q, \text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
and $tcorr: \forall k < k_tm \ M. \text{ar_tape_correspondence } (\Gamma_tm \ M) (\text{le_tm } M)$
 $(\text{bl_tm } M)$
 $(\text{mt_tape } cM \ k) (\text{mt_tape } c0 \ k)$
and $ppos: \forall k < k_tm \ M. \text{mt_pos } c0 \ k = (\text{if } \text{mt_pos } cM \ k = 0 \text{ then } \text{Suc } 0$
 $\text{else } \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos } cM \ k)$
 $+ \text{block_width } (\Gamma_tm \ M))$
and $poskle: \forall k < k_tm \ M. \text{posk } k = \text{AR_AtLE} \longleftrightarrow \text{mt_pos } cM \ k = 0$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\}$
and $\text{pad0}: \forall j \geq k_tm \ M. \text{mt_tape } c0 \ j (\text{mt_pos } c0 \ j) = \text{BLANK4}$
and $\text{src0}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $j \leq k_tm \ M - 1 \implies$
 $(\exists c \ m. (c0, c) \in (\text{mttm_step } (\text{ar_delta_write } M)) \wedge^m$
 $\wedge m \leq j * (2 * \text{block_width } (\Gamma_tm \ M))$
 $\wedge \text{mt_state } c = (q, \text{AR_SimWrite}, k_unidx \ j, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c = (\lambda k. \text{if } k_idx \ k < j$
 $\text{then } (\text{if } \text{mt_pos } cM \ k = 0 \text{ then } \text{mt_tape } c0 \ k$
 $\text{else } (\lambda \text{pos}. \text{if } \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos } cM \ k)$
 $\leq \text{pos}$
 $\wedge \text{pos} < \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos}$
 $cM \ k)$
 $+ \text{block_width } (\Gamma_tm \ M)$
 $\text{then } \text{write_bit } (\Gamma_tm \ M) (\text{bl_tm } M) (\text{buf } k)$
 $(\text{pos} - \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos}$
 $cM \ k))$
 $\text{else } \text{mt_tape } c0 \ k \ \text{pos}))$
 $\text{else } \text{mt_tape } c0 \ k)$
 $\wedge \text{mt_pos } c = \text{mt_pos } c0)$
 $\langle \text{proof} \rangle$

The full write phase, re-mirrored into $\text{mttm_step } (\text{ar_delta_write } M)$: the prefix walk $(\text{ar_write_prefix_in_sub})$ over the first $k_tm \ M - 1$ tapes followed by the unified last-tape write $(\text{ar_write_tape_finish_step_in_sub})$, landing at AR_SimAdvance with every proper tape's block overwritten by $\text{write_bit } (\text{buf } k)$ and every LE tape / head unchanged. Aggregate cost $\leq k_tm \ M \cdot 2b$. The split-to-full descriptor collapse, the cardinality bookkeeping, and the cost bound are relation-agnostic and transfer verbatim; only the relation and the two helper references change.

lemma $\text{ar_write_phase_in_sub}$:
fixes $M :: ('q, 'a) \text{ mttm}$
and $cM :: ('a, 'q) \text{ mt_config}$

and $c0 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $kge2: 2 \leq \text{block_width } (\Gamma_tm \ M)$
and $stg0: \text{mt_state } c0 = (q, \text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
and $tcorr: \forall k < k_tm \ M. \text{ar_tape_correspondence } (\Gamma_tm \ M) (\text{le_tm } M)$
 $(\text{bl_tm } M)$
 $(\text{mt_tape } cM \ k) (\text{mt_tape } c0 \ k)$
and $ppos: \forall k < k_tm \ M. \text{mt_pos } c0 \ k = (\text{if } \text{mt_pos } cM \ k = 0 \text{ then } \text{Suc } 0$
 $\text{else } \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos } cM \ k)$
 $+ \text{block_width } (\Gamma_tm \ M))$
and $poskle: \forall k < k_tm \ M. \text{posk } k = \text{AR_AtLE} \longleftrightarrow \text{mt_pos } cM \ k = 0$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\}$
and $\text{pad0}: \forall j \geq k_tm \ M. \text{mt_tape } c0 \ j (\text{mt_pos } c0 \ j) = \text{BLANK4}$
and $\text{src0}: \text{ar_stage_bounded } (\text{bl_tm } M) (k_tm \ M)$
 $(\text{AR_SimWrite}, 0, 0, \text{buf}, \text{dvec}, \text{posk})$
shows $\exists c \ m. (c0, c) \in (\text{mttm_step } (\text{ar_delta_write } M)) \wedge m$
 $\wedge m \leq k_tm \ M * (2 * \text{block_width } (\Gamma_tm \ M))$
 $\wedge \text{mt_state } c = (q, \text{AR_SimAdvance}, k_unidx \ 0, 0, \text{buf}, \text{dvec}, \text{posk})$
 $\wedge \text{mt_tape } c = (\lambda k. \text{if } k < k_tm \ M$
 $\text{then } (\text{if } \text{mt_pos } cM \ k = 0 \text{ then } \text{mt_tape } c0 \ k$
 $\text{else } (\lambda \text{pos}. \text{if } \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos } cM \ k)$
 $\leq \text{pos}$
 $\wedge \text{pos} < \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos}$
 $cM \ k)$
 $+ \text{block_width } (\Gamma_tm \ M)$
 $\text{then } \text{write_bit } (\Gamma_tm \ M) (\text{bl_tm } M) (\text{buf } k)$
 $(\text{pos} - \text{sim_pos } (\text{block_width } (\Gamma_tm \ M)) (\text{mt_pos}$
 $cM \ k))$
 $\text{else } \text{mt_tape } c0 \ k \ \text{pos}))$
 $\text{else } \text{mt_tape } c0 \ k)$
 $\wedge \text{mt_pos } c = \text{mt_pos } c0$
 $\langle \text{proof} \rangle$

The advance left-walk loop, re-mirrored into mttm_step ($\text{ar_delta_advance } M$). From an AR_SimAdvance stage at bit-counter 0, $n \ M'$ -steps walk the head n cells L (counter $0 \rightarrow n$), tape and other heads unchanged. The $\text{relpow_invariant_chain}$ loop, the per-step $\neq \text{LE4}$ guard, and the displacement bound (ar_disp_le_2k) are relation-agnostic and transfer verbatim; only the relation and the one helper reference change.

lemma $\text{ar_advance_walk_loop_in_sub}$:

fixes $M :: ('q, 'a) \text{ mttm}$
and $c0 :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes $vM: \text{valid_mttm } M$
and $qQ: q \in Q_tm \ M$
and $\text{stg}: \text{mt_state } c0 = (q, \text{AR_SimAdvance}, tk, 0, \text{buf}, \text{dvec}, \text{posk})$
and $\text{buf_valid}: \forall k. \text{buf } k \in \Gamma_tm \ M \cup \{\text{bl_tm } M\}$
and $\text{ndisp}: n < \text{ar_disp } (\text{block_width } (\Gamma_tm \ M)) (\text{dvec } tk) (\text{posk } tk)$
and $\text{notLE}: \bigwedge j. j < n \implies \text{mt_tape } c0 \ tk (\text{mt_pos } c0 \ tk - j) \neq \text{LE4}$

and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in (mttm_step\ (ar_delta_advance\ M)) \wedge \wedge n$
 $\wedge mt_state\ c' = (q, AR_SimAdvance, tk, n, buf, dvec, posk)$
 $\wedge mt_pos\ c'\ tk = mt_pos\ c0\ tk - n$
 $\wedge mt_tape\ c' = mt_tape\ c0$
 $\wedge (\forall k'. k' \neq tk \longrightarrow mt_pos\ c'\ k' = mt_pos\ c0\ k')$
 $\langle proof \rangle$

The unified single-tape advance, non-last tape, re-mirrored into $mttm_step\ (ar_delta_advance\ M)$: the R /non- R dispatch. An R -move needs no head motion (single boundary step, cost 1); a non- R -move walks the head $D = ar_disp$ cells L (the $D - 1$ -step $ar_advance_walk_loop_in_sub$ then the final boundary L -move), landing at $start - D$. The walk-then-boundary decomposition and the conditional head conclusion transfer verbatim; only the relation and the two helper references change.

lemma $ar_advance_tape_step_in_sub$:
fixes $M :: ('q, 'a)\ mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $notlast: \neg is_last_k\ M\ tk$
and $stg: mt_state\ c0 = (q, AR_SimAdvance, tk, 0, buf, dvec, posk)$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $dge1: dvec\ tk \neq dir.R$
 $\implies 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$
and $notLE: \bigwedge m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$
 $\implies mt_tape\ c0\ tk\ (mt_pos\ c0\ tk - m) \neq LE4$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in (mttm_step\ (ar_delta_advance\ M))$
 $\wedge \wedge (if\ dvec\ tk = dir.R\ then\ 1$
 $\quad else\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk))$
 $\wedge mt_state\ c' = (q, AR_SimAdvance, k_succ\ tk, 0, buf, dvec,$
 $\quad posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk)))$
 $\wedge mt_tape\ c' = mt_tape\ c0$
 $\wedge mt_pos\ c' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c0$
 $\quad else\ (mt_pos\ c0)(tk := mt_pos\ c0\ tk$
 $\quad - ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk$
 $\quad tk)))$
 $\langle proof \rangle$

The unified single-tape advance, last tape, re-mirrored into $mttm_step\ (ar_delta_advance\ M)$: as $ar_advance_tape_step_in_sub$ but tk is the last tape, so the boundary step ($ar_advance_finish_step_in_sub$) transitions to $AR_SimNext$ (current-tape field reset to $k_unidx\ 0$) rather than advancing

to $k_succ\ tk$. Same R/non-R dispatch and walk-then-boundary decomposition; only the relation and the two helper references change.

lemma $ar_advance_tape_finish_step_in_sub$:
fixes $M :: ('q, 'a)\ mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $last: is_last_k\ M\ tk$
and $stg: mt_state\ c0 = (q, AR_SimAdvance, tk, 0, buf, dvec, posk)$
and $buf_valid: \forall k. buf\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $dge1: dvec\ tk \neq dir.R$
 $\implies 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$
and $notLE: \bigwedge m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk)$
 $\implies mt_tape\ c0\ tk\ (mt_pos\ c0\ tk - m) \neq LE4$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, tk, 0, buf, dvec, posk)$
shows $\exists c'. (c0, c') \in (mttm_step\ (ar_delta_advance\ M))$
 $\wedge (if\ dvec\ tk = dir.R\ then\ 1$
 $\quad else\ ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk\ tk))$
 $\wedge\ mt_state\ c' = (q, AR_SimNext, k_unidx\ 0, 0, buf, dvec,$
 $\quad posk(tk := ar_newpos\ (dvec\ tk)\ (posk\ tk)))$
 $\wedge\ mt_tape\ c' = mt_tape\ c0$
 $\wedge\ mt_pos\ c' = (if\ dvec\ tk = dir.R\ then\ mt_pos\ c0$
 $\quad else\ (mt_pos\ c0)(tk := mt_pos\ c0\ tk$
 $\quad - ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ tk)\ (posk$
 $\quad tk)))$
 $\langle proof \rangle$

The advance prefix walk, re-mirrored into $mttm_step\ (ar_delta_advance\ M)$: a custom induction on the tape index j ($j \leq k_tm\ M - 1$, every tape it touches non-last), iterating $ar_advance_tape_step_in_sub$ from the boundary tape $k_unidx\ 0$. The tape is constant; the carried state is a $k_idx\ k < j$ split over the $posk$ and position vectors. The induction, the descriptor collapse, and the per-tape $notLE/dge1$ entry facts are relation-agnostic and transfer verbatim; only the relation and the one helper reference change.

lemma $ar_advance_prefix_in_sub$:
fixes $M :: ('q, 'a)\ mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0: mt_state\ c0 = (q, AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
and $buf_valid: \forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $dge1: \forall k < k_tm\ M. dvec\ k \neq dir.R$
 $\implies 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ k)\ (posk0\ k)$
and $notLE: \forall k < k_tm\ M. \forall m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))$

$(dvec\ k)\ (posk0\ k)$
 $\longrightarrow mt_tape\ c0\ k\ (mt_pos\ c0\ k - m) \neq LE4$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
shows $j \leq k_tm\ M - 1 \implies$
 $(\exists c\ m. (c0, c) \in (mttm_step\ (ar_delta_advance\ M)) \wedge m$
 $\wedge m \leq j * (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c = (q, AR_SimAdvance, k_unidx\ j, 0, buf0, dvec,$
 $(\lambda k. \text{if } k_idx\ k < j \text{ then } ar_newpos\ (dvec\ k)\ (posk0\ k)$
 $\text{else } posk0\ k))$
 $\wedge mt_tape\ c = mt_tape\ c0$
 $\wedge mt_pos\ c = (\lambda k. \text{if } k_idx\ k < j$
 $\text{then } (\text{if } dvec\ k = dir.R \text{ then } mt_pos\ c0\ k$
 $\text{else } mt_pos\ c0\ k$
 $\text{--- } ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ k)\ (posk0\ k))$
 $\text{else } mt_pos\ c0\ k))$
 $\langle proof \rangle$

The full advance phase, re-mirrored into $mttm_step\ (ar_delta_advance\ M)$: the prefix walk ($ar_advance_prefix_in_sub$) over the first $k_tm\ M - 1$ tapes followed by the unified last-tape advance ($ar_advance_tape_finish_step_in_sub$), landing at $AR_SimNext$ (current-tape field $k_unidx\ 0$) with every head moved to M 's new position and every $posk$ updated by ar_newpos . The tape is unchanged. Aggregate cost $\leq k_tm\ M \cdot 2b$. The split-to-full descriptor collapse and cost bound are relation-agnostic and transfer verbatim; only the relation and the two helper references change.

lemma $ar_advance_phase_in_sub$:
fixes $M :: ('q, 'a)\ mttm$
and $c0 :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes $vM: valid_mttm\ M$
and $qQ: q \in Q_tm\ M$
and $kge2: 2 \leq block_width\ (\Gamma_tm\ M)$
and $stg0: mt_state\ c0 = (q, AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
and $buf_valid: \forall k. buf0\ k \in \Gamma_tm\ M \cup \{bl_tm\ M\}$
and $dge1: \forall k < k_tm\ M. dvec\ k \neq dir.R$
 $\longrightarrow 0 < ar_disp\ (block_width\ (\Gamma_tm\ M))\ (dvec\ k)\ (posk0\ k)$
and $notLE: \forall k < k_tm\ M. \forall m. m < ar_disp\ (block_width\ (\Gamma_tm\ M))$
 $(dvec\ k)\ (posk0\ k)$
 $\longrightarrow mt_tape\ c0\ k\ (mt_pos\ c0\ k - m) \neq LE4$
and $pad0: \forall j \geq k_tm\ M. mt_tape\ c0\ j\ (mt_pos\ c0\ j) = BLANK4$
and $src0: ar_stage_bounded\ (bl_tm\ M)\ (k_tm\ M)$
 $(AR_SimAdvance, 0, 0, buf0, dvec, posk0)$
shows $\exists c\ m. (c0, c) \in (mttm_step\ (ar_delta_advance\ M)) \wedge m$
 $\wedge m \leq k_tm\ M * (2 * block_width\ (\Gamma_tm\ M))$
 $\wedge mt_state\ c = (q, AR_SimNext, k_unidx\ 0, 0, buf0, dvec,$
 $(\lambda k. \text{if } k < k_tm\ M \text{ then } ar_newpos\ (dvec\ k)\ (posk0\ k)$

```

      else posk0 k))
    ∧ mt_tape c = mt_tape c0
    ∧ mt_pos c = (λk. if k < k_tm M
      then (if dvec k = dir.R then mt_pos c0 k
        else mt_pos c0 k - ar_disp (block_width (Γ_tm M)) (dvec k)
      )
    )
    (posk0 k))
    else mt_pos c0 k)
  ⟨proof⟩

```

3.17 Walker preservation — per-substep M'-step lemmas

One preservation lemma per substep predicate: an M' -step out of a config satisfying the current invariant lands at a config satisfying the next invariant in the cycle (or splits the disjunction when the substep's relation has multiple destination arms). Together with the chunked engine, these characterise the walker's per-step evolution: the substep idx carried in the M' -config is the dispatch discriminator at each step, no chain pinning needed.

```

lemma ar_walker_step_from_boundary:
  fixes M :: ('q, 'a) mttm
    and cM :: ('a, 'q) mt_config
    and c' c'' :: (sym4, 'q × 'a ar_stage) mt_config
  assumes inv: ar_walker_at_boundary M cM c'
    and step: (c', c'') ∈ mttm_step (alphabet_reduce_delta M)
  shows ar_walker_in_read M cM c'' ∨ ar_walker_at_compute M cM c''
  ⟨proof⟩

```

```

lemma ar_walker_step_from_in_read:
  fixes M :: ('q, 'a) mttm
    and cM :: ('a, 'q) mt_config
    and c' c'' :: (sym4, 'q × 'a ar_stage) mt_config
  assumes inv: ar_walker_in_read M cM c'
    and step: (c', c'') ∈ mttm_step (alphabet_reduce_delta M)
  shows ar_walker_in_read M cM c'' ∨ ar_walker_at_compute M cM c''
  ⟨proof⟩

```

```

lemma ar_walker_step_from_at_compute:
  fixes M :: ('q, 'a) mttm
    and cM :: ('a, 'q) mt_config
    and c' c'' :: (sym4, 'q × 'a ar_stage) mt_config
  assumes inv: ar_walker_at_compute M cM c'
    and step: (c', c'') ∈ mttm_step (alphabet_reduce_delta M)
    and vM: valid_mttm M
    and qQ: mt_state cM ∈ Q_tm M
    and kge2: 2 ≤ block_width (Γ_tm M)
    and tapeG: ∧k. mt_tape cM k (mt_pos cM k) ∈ Γ_tm M
  shows ar_walker_in_write M cM c''
  ⟨proof⟩

```

lemma *ar_walker_step_from_in_write*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $cM :: ('a, 'q) \text{ mt_config}$
and $c' c'' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes *inv*: $\text{ar_walker_in_write } M \text{ } cM \text{ } c'$
and *step*: $(c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
shows $\text{ar_walker_in_write } M \text{ } cM \text{ } c'' \vee \text{ar_walker_in_advance } M \text{ } cM \text{ } c''$
 $\langle \text{proof} \rangle$

lemma *ar_walker_step_from_in_advance*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $cM :: ('a, 'q) \text{ mt_config}$
and $c' c'' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes *inv*: $\text{ar_walker_in_advance } M \text{ } cM \text{ } c'$
and *step*: $(c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
shows $\text{ar_walker_in_advance } M \text{ } cM \text{ } c'' \vee \text{ar_walker_at_next } M \text{ } cM \text{ } c''$
 $\langle \text{proof} \rangle$

The sixth and final substep preservation closes the walker suite mechanically: an M' -step out of $AR_SimNext$ lands at $AR_SimRead$ (cycle close), $AR_HaltAccept$, or $AR_HaltReject$ by *ar_step_from_SimNext_lands*. This is a pure dispatch lemma — establishing the boundary for the reconstructed M -successor at a cycle close is a separate cycle-level concern handled by *ar_walker_cycle_close* and the chunked reverse engine. Keeping the *at_next* walker preservation thin keeps the suite uniform — all six are one-substep mechanical lemmas. Itself currently uncalled — the cycle close runs through *ar_walker_cycle_close* directly — kept to complete the six-member walker-preservation suite.

lemma *ar_walker_step_from_at_next*:
fixes $M :: ('q, 'a) \text{ mttm}$
and $cM :: ('a, 'q) \text{ mt_config}$
and $c' c'' :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$
assumes *inv*: $\text{ar_walker_at_next } M \text{ } cM \text{ } c'$
and *step*: $(c', c'') \in \text{mttm_step } (\text{alphabet_reduce_delta } M)$
shows $\text{fst } (\text{snd } (\text{mt_state } c'')) = AR_SimRead$
 $\vee \text{fst } (\text{snd } (\text{mt_state } c'')) = AR_HaltAccept$
 $\vee \text{fst } (\text{snd } (\text{mt_state } c'')) = AR_HaltReject$
 $\langle \text{proof} \rangle$

3.18 Cycle-close bridging lemma

An $AR_SimNext$ step closes a simulation cycle: the walker is re-established at a boundary for the next M -configuration. The successor is *reconstructed* from the pinned compute branch (the $\exists cM'$ in the conclusion): under non-deterministic $\text{delta_tm } M$ the source cM may have several successors, so the fired branch is recovered from the walker's own witness chain rather than assumed. The forward arm is rebuilt entirely over the substep sub-relations and

each walker witness chain is pinned by a *chain_ar_delta_X_to_Y_uniq* suite.

The closing next step is inverted by *ar_next_step_inv*, which dispatches on the reconstructed *M*-state q' into three outcomes: continue ($q' \notin \{t, r\}$, landing at *AR_SimRead*), accept ($q' = t_tm\ M$) or reject ($q' = r_tm\ M$). All three re-establish *ar_walker_at_boundary M cMn c''*: the boundary invariant carries the halt cases too — *ar_simulates*'s state disjunction has dedicated accept/reject arms, and both *ar_posk_consistent* and *ar_at_read_boundary* are *AR_SimRead*-guarded, hence vacuous off the read boundary. The three predicates are discharged from the explicit endpoint by the shared semantic lemmas (*ar_write_tape_correspondence*, *ar_advance_newsimpos*, *ar_newpos_atLE_iff*). No union chain is pinned and *ar_simulates_forward_step* is not invoked.

lemma *ar_walker_cycle_close*:
fixes $M :: ('q, 'a)\ mttm$
and $w :: 'a\ list$
and $cM :: ('a, 'q)\ mt_config$
and $c'\ c'' :: (sym4, 'q \times 'a\ ar_stage)\ mt_config$
assumes *inv*: *ar_walker_at_next M cM c'*
and *step*: $(c', c'') \in mttm_step\ (alphabet_reduce_delta\ M)$
and *vM*: *valid_mttm M*
and *qQ*: $mt_state\ cM \in Q_tm\ M$
and *kge2*: $2 \leq block_width\ (\Gamma_tm\ M)$
and *tapeG*: $\bigwedge k. mt_tape\ cM\ k\ (mt_pos\ cM\ k) \in \Gamma_tm\ M$
and *w_sub*: $set\ w \subseteq Sigma_tm\ M$
and *reach_M*: $(init_config_mttm\ M\ w, cM) \in (mttm_step\ (delta_tm\ M))^*$
and *lebl*: $le_tm\ M \neq bl_tm\ M$
shows $\exists cM'. (cM, cM') \in mttm_step\ (delta_tm\ M)$
 $\wedge ar_walker_at_boundary\ M\ cM'\ c''$
 $\langle proof \rangle$

3.19 Chunked reverse engine

The reverse-arm loop invariant: the substep-walker is in *some* stage of the cycle. Six disjuncts, one per walker predicate. This is the AR analogue of carrying *ae_simulates* across AE's reverse induction, but stage-granular: where AE peels a whole fixed-length cycle per *ae_backward_stage*, AR peels one *M'*-substep and dispatches on the current substep tag (the cycle length is data-dependent here, which is why the pivot to the substep walker was needed in the first place).

definition *ar_walker* ::
 $('q, 'a)\ mttm$
 $\Rightarrow ('a, 'q)\ mt_config$
 $\Rightarrow (sym4, 'q \times 'a\ ar_stage)\ mt_config \Rightarrow bool$ **where**
 $ar_walker\ M\ cM\ c' \longleftrightarrow$
 $ar_walker_at_boundary\ M\ cM\ c'$

$\vee \text{ ar_walker_in_read } M \text{ cM } c'$
 $\vee \text{ ar_walker_at_compute } M \text{ cM } c'$
 $\vee \text{ ar_walker_in_write } M \text{ cM } c'$
 $\vee \text{ ar_walker_in_advance } M \text{ cM } c'$
 $\vee \text{ ar_walker_at_next } M \text{ cM } c'$

Chunked-induction engine for the reverse arm, the AR analogue of AE's *ae_simulation_phase_chunked_reverse*. Given a finite M' -trace (c', c_acc) of length m ending in the canonical accept config $(t_tm \ M, \text{ ar_accept_stage } (bl_tm \ M))$ and the walker invariant at c' relative to a reachable cM , deliver an accepting M -path from cM . Strong induction on m ; at each level dispatch on the walker stage and peel one M' -step:

- the five active stages (*in_read*, *at_compute*, *in_write*, *in_advance*, *at_next*, and an active read boundary) all have $c' \neq c_acc$, so $m = \text{Suc } m'$; peel the step via the matching preservation lemma (*at_compute* emits one M -step into the *in_write* witness; *at_next* emits one via *ar_walker_cycle_close* and advances cM), then recurse on m' ;
- a boundary at the accept config closes the induction: *ar_simulates_accept_iff* forces $mt_state \ cM = t_tm \ M$;
- a boundary at the reject config is impossible — the trace runs to the accept config, but reject is terminal (*ar_reject_terminal*) and the reject config is not the accept config.

The conclusion is in *rtrancl* form for the downstream *Lang_mttm*-repacking in *alphabet_reduce_language*.

lemma *ar_simulation_phase_chunked_reverse*:

fixes $M :: ('q, 'a) \text{ mttm}$

and $w :: 'a \text{ list}$

and $cM :: ('a, 'q) \text{ mt_config}$

and $c' \ c_acc :: (\text{sym4}, 'q \times 'a \text{ ar_stage}) \text{ mt_config}$

and $m :: \text{nat}$

assumes vM : *valid_mttm* M

and w_sub : $\text{set } w \subseteq \text{Sigma_tm } M$

and $card_ge$: $\text{card } (\Gamma_tm \ M) \geq 4$

and $lebl$: $le_tm \ M \neq bl_tm \ M$

and $reach_M$: $(\text{init_config_mttm } M \ w, cM) \in (\text{mttm_step } (\text{delta_tm } M))^*$

and $trace$: $(c', c_acc) \in (\text{mttm_step } (\text{alphabet_reduce_delta } M))^{\wedge\wedge m}$

and $accept$: $mt_state \ c_acc = (t_tm \ M, \text{ ar_accept_stage } (bl_tm \ M))$

and $walk$: *ar_walker* $M \ cM \ c'$

shows $\exists cM_final. (cM, cM_final) \in (\text{mttm_step } (\text{delta_tm } M))^*$

$\wedge mt_state \ cM_final = t_tm \ M$

<proof>

Classical language-equivalence (biconditional) for alphabet reduction: an $'a$ -word w over the input alphabet is in M 's language iff its *sym4*-encoding

is in the language of the reduced machine M' . Pairs the forward inclusion *alphabet_reduce_language_forward* (in *AlphabetReduction_Theorems.thy*) with the reverse leg supplied by *ar_simulation_phase_chunked_reverse*.

Unlike AE, the alphabet-reduction construction has no input validation prefix (the no-validation design point), so the reverse direction needs neither a validation-peel nor a determinism / chain-uniqueness argument: the encoded input's accepting M' -run is handed to the engine directly off the initial read-boundary correspondence (*ar_walker_at_boundary*, the conjunction of the three *_init* invariants). No *det_mttm* hypothesis is required.

The *set* $w \subseteq \text{Sigma_tm } M$ guard is required (and matches AE's statement): *Lang_mttm* bakes in the input-alphabet restriction, so $w \in \text{Lang_mttm } M$ supplies the guard for free in the forward direction, but membership of the encoded word in *Lang_mttm* M' says nothing about w 's alphabet. For w outside the input alphabet the encoded word can still be accepted by M' while $w \notin \text{Lang_mttm } M$.

theorem *alphabet_reduce_language:*

fixes $M :: ('q, 'a) \text{ mttm}$

assumes vM : *valid_mttm* M

and s_neq_t : $s_tm\ M \neq t_tm\ M$

and s_neq_r : $s_tm\ M \neq r_tm\ M$

and le_neq_bl : $le_tm\ M \neq bl_tm\ M$

and $card_ge$: $card\ (\Gamma_tm\ M) \geq 4$

shows $\forall w. \text{set } w \subseteq \text{Sigma_tm } M$

$\longrightarrow (\text{encode_input_ar } (\Gamma_tm\ M) (bl_tm\ M) w \in \text{Lang_mttm}$
 $(\text{alphabet_reduce } M$
 $:: ('q \times 'a \text{ ar_stage, sym4}) \text{ mttm}))$
 $= (w \in \text{Lang_mttm } M)$

<proof>

end

References

- [1] F. J. Balbach. The Cook-Levin theorem. *Archive of Formal Proofs*, January 2023. https://isa-afp.org/entries/Cook_Levin.html, Formal proof development.
- [2] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.
- [3] C. Dalvit and R. Thiemann. A verified translation of multitape Turing machines into singletape Turing machines. *Archive of Formal Proofs*, November 2022. https://isa-afp.org/entries/Multitape_To_Singletape_TM.html, Formal proof development.

- [4] P. C. Fischer. Multi-tape and infinite-state automata—a survey. *Commun. ACM*, 8(12):799–805, Dec. 1965.
- [5] J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, 1969.
- [6] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [7] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 1–66. Elsevier, 1990.
- [8] J. Xu, X. Zhang, C. Urban, S. J. C. Joosten, and F. Regensburger. Universal Turing machine. *Archive of Formal Proofs*, February 2019. https://isa-afp.org/entries/Universal_Turing_Machine.html, Formal proof development.