Three Counterexamples and One-Sentence Corrections for IETF Internet-Draft draft-ietf-core-href-30, with a Lean Proof for the Path Rule

Nobuo Otoi
BA Cybersecurity, Tokyo, Japan
otoi@ba-cybersecurity.com

October 2026 (version 1)

Abstract

draft-ietf-core-href-30 defines Constrained Resource Identifiers (CRIs), a CBOR form of URIs, with a procedure that resolves a CRI *reference* against a base CRI (Section 5.3) and a procedure that converts a CRI reference to a URI reference (Section 6.1), which the draft calls the equivalent URI reference. For a base and a reference we compare two computations: the URI of the Section 5.3 result, and the RFC 3986 resolution of the converted reference against the base. We give three inputs on which the two differ, with every step written out. (A) Resolving `["t", null, ["p"]]` (the CRI for `t:/p`) against `s://h` yields `t://h/p` by Section 5.3, whereas Section 2.1 assigns no authority to the equivalent URI reference and RFC 3986 gives a reference with a scheme its own authority; the same input, used as a reference to itself, also fails the draft's requirement that resolution yield the original CRI. (B) The reference `[1]`, which removes the last path segment of the base, is converted to the empty URI reference, which denotes the base itself. (C) The reference `[1, ["", "a"]]` is converted to `/a`, an absolute path, whereas Section 5.3 gives `/b//a` for the base path `/b/c`. We propose one sentence of correction for each. On an enumeration of 720,120 base–reference pairs with an authority, the two computations differ on 93,810; after the corrections they differ on none of the 713,660 pairs that remain convertible (the corrections refuse 10,540 pairs and admit 4,080 that the draft's syntax check rejects). In Lean 4 (core library; axioms `propext` and `Quot.sound` only; no `sorry`) the counterexamples are checked by kernel evaluation, and the corrected conversion of a discard-form reference, resolved per RFC 3986, is proved equal to the Section 5.3 resolution for every base with an authority and paths of any length. We make no claim about bases without an authority.

1 Introduction

A CRI [1] carries the five components of a URI [2] as a CBOR [3] array. A *CRI reference* is the compact form for relative links: instead of the leading dot segments and slash of a URI reference, it carries a *discard* value that says how many trailing segments of the base path to remove. Two procedures on references are normative. Section 5.3 resolves a reference against a base CRI ("An application MUST implement this operation by applying the algorithm specified in Section 5.3 (or any algorithm that is functionally equivalent to it)", Section 5), and Section 6.1 converts a CRI reference to a URI reference ("Applications MUST convert a CRI reference to a URI reference by determining the components ... according to the following steps"). The draft tabulates the correspondence between discard values and URI references (its Table 1: `[1, ["a"]]` is `a`, `[2, ["a"]]` is `../a`, `[true, ["a"]]` is `/a`).

This note compares, for a base and a reference, the URI of the Section 5.3 result with the RFC 3986 resolution of the Section 6.1 conversion against the base (Section 3 defines the comparison), and reports three inputs on which the two differ. Each is a finite computation

that the reader can redo from the quoted text in a few minutes (Section 3). We propose a one-sentence correction for each (Section 4) and support the corrections in two ways (Section 5): an exhaustive enumeration of small inputs, and a Lean 4 proof that the corrected conversion agrees with RFC 3986 resolution for all path lengths. Section 7 states exactly what is and is not claimed.

2 The text under test

RFC 3986. Section 5.2.2 resolves a reference R against a base B (which has no fragment, Section 5.1). If R has a scheme, the target is R with dot segments removed and in particular has R's authority (possibly none). Otherwise the scheme is B's; if R has an authority, authority and path are R's; otherwise the authority is B's and the path is B's path when R's path is empty, $\mathrm{rds}(R.\mathit{path})$ when it starts with "/", and $\mathrm{rds}(\mathrm{merge}(B,R))$ otherwise, where merge appends R's path to B's path with its last segment removed (or to "/" when B has an authority and an empty path). The query is R's, except that an empty-path reference without a query keeps B's query; the fragment is R's.

Notation. We write a path as a list of segments $p = [p_1, \ldots, p_k]$, $|p| = k$, with concatenation $p +\!\!+ q$, $\mathrm{take}_j(p) = [p_1, \ldots, p_{\min(j,k)}]$ and $\mathrm{dropLast}(p) = \mathrm{take}_{k-1}(p)$; ε is the empty segment. For a base B with an authority and path p_B, Sections 5.2.3 and 5.2.4 of RFC 3986 become, on segment lists,

$$\mathrm{merge}_B(p) = \begin{cases} p & \text{if } p_B = [] \\ \mathrm{dropLast}(p_B) +\!\!+ p & \text{otherwise,} \end{cases} \qquad \mathrm{rds}(p) = R([], p),$$

$$R(o, []) = o,$$

$$R(o, \text{“.”} :: t) = \begin{cases} o +\!\!+ [\varepsilon] & t = [] \\ R(o,t) & t \neq [], \end{cases}$$

$$R(o, \text{“..”} :: t) = \begin{cases} \mathrm{dropLast}(o) +\!\!+ [\varepsilon] & t = [] \\ R(\mathrm{dropLast}(o), t) & t \neq [], \end{cases}$$

$$R(o, s :: t) = R(o +\!\!+ [s], t) \quad \text{for any other segment } s.$$

A URI reference without scheme or authority is $u = (\mathit{rooted}, p, q, f)$ with p a segment list or $\bot$ (the empty path) and q, f possibly $\bot$. Section 5.2.2 then reads

$$\rho_{3986}(B,u).\mathit{path} = \begin{cases} p_B & p = \bot \\ \mathrm{rds}(p) & \mathit{rooted} \\ \mathrm{rds}(\mathrm{merge}_B(p)) & \text{otherwise,} \end{cases} \qquad \rho_{3986}(B,u).\mathit{query} = \begin{cases} q & q \neq \bot \\ q_B & q = \bot,\ p = \bot \\ [] & q = \bot,\ p \neq \bot, \end{cases}$$

and the fragment is f. (The string form of a rooted p is "/"p_1"/"$\cdots p_k$; of an unrooted one, p_1"/"$\cdots p_k$; $\bot$ is the empty string.) These definitions are the ones formalized in Lean (Section 5); their agreement with the string algorithms of RFC 3986 is checked in Section 5.

The CRI data model. A CRI reference (Figure 2 of [1], in CDDL [4]) is an array whose head is either a scheme and an authority or a discard value, followed by path, query and fragment. The authority is a host, or one of two "no authority" values: null for a root-based path (`a:/p`) and true for a rootless path (`a:p`). Table 2 of Section 5.1 gives null as the default value of the authority section. Section 2.1 says of references:

> (For example, for a CRI reference where authority is either not set or has either of the NOAUTHORITY values, the equivalent URI reference's authority is absent.)

Path segments are never "." or ".." (constraint C9).

Section 5.3 (resolution). Steps 3 and 5 read:

> 3. If the value of discard is true in the CRI reference (which is implicitly the case when scheme and/or authority are present in the reference), replace the path in the buffer with the empty array, set query to empty and fragment to null, and set a true authority to null. If the value of discard is an unsigned integer, remove as many elements from the end of the path array; if it is non-zero, set query to empty and fragment to null. Set discard to true in the buffer.
>
> 5. If query is non-null in the CRI reference, set fragment to null in the buffer. Apart from the path and discard, copy all non-null sections from the CRI reference to the buffer in sequence.

Step 4 reads: "If the path section is non-null in the CRI reference, append all elements from the path array to the array in the path section in the buffer; set query to empty and fragment to null." For a reference in discard form $r = (d, \pi, \kappa, \varphi)$ ($d \in \{\mathit{true}\} \cup \mathbb{N}$; π a segment list or $\bot$; κ a query or $\bot$; φ a fragment or $\bot$) and a base B with path p_B and query q_B, steps 3–5 compute

$$P_3 = \begin{cases} [] & d = \mathit{true} \\ \mathrm{take}_{|p_B|-n}(p_B) & d = n, \end{cases} \qquad Q_3 = \begin{cases} q_B & d = 0 \\ [] & \text{otherwise}, \end{cases}$$

$$\rho_{5.3}(B,r).\mathit{path} = \begin{cases} P_3 +\!\!+ \pi & \pi \neq \bot \\ P_3 & \pi = \bot, \end{cases} \qquad \rho_{5.3}(B,r).\mathit{query} = \begin{cases} \kappa & \kappa \neq \bot \\ [] & \kappa = \bot,\ \pi \neq \bot \\ Q_3 & \kappa = \bot,\ \pi = \bot, \end{cases}$$

and $\rho_{5.3}(B,r).\mathit{fragment} = \varphi$.

Section 6.1 (conversion). The path rule reads, in part:

> If the CRI reference contains a discard item of value true, the path component is considered rooted. If it contains a discard item of value 0 and the path item is present, the conversion fails. If it contains a positive discard item, the path component is considered unrooted and prefixed by as many "../" components as the discard value minus one indicates. If the discard value is 1 and the first element of the path contains a :, the path component is prefixed by "./". ... If the CRI reference contains one or more path items, the path component is constructed by concatenating the sequence of representations of these items. These representations generally contain a leading slash ("/") character and the value of each item, processed as discussed below. The leading slash character is omitted for the first path item only if the path component is considered "unrooted".

The rule ends with syntax checks: when neither authority nor scheme is present, the path component must match path-absolute, path-noscheme or path-empty of RFC 3986, else the conversion fails. In the notation above, with $\mathit{items} = \pi$ (or [] if $\pi = \bot$), the path part of $\mathrm{conv}(r)$ is

$$\begin{aligned} d = \mathit{true}:&\ (\mathit{rooted},\ \mathit{items}); \\ d = 0:&\ \text{fail if } \pi \neq \bot, \text{ else the empty path}; \\ d = n \geq 1:&\ (\mathit{unrooted},\ \mathit{pre} +\!\!+ \mathit{items}), \qquad \mathit{pre} = \begin{cases} [\text{“.”}] & n = 1 \text{ and “:” } \in \mathit{items}_1 \\ [\text{“..”}]^{n-1} & n \geq 2 \\ [] & \text{otherwise}, \end{cases} \end{aligned}$$

followed by the string rendering, and the query part is κ if $\kappa \neq \bot$ and $\kappa \neq []$, else $\bot$. Section 2.3 states that some well-formed references cannot be converted, naming `[0, ["p"]]` and `[0, null, []]`.

3 Three counterexamples

For a base B and a reference r we compute two things: u_1, the URI of the Section 5.3 result (Section 6.1 applied to the resolved full CRI); and u_2, the RFC 3986 resolution, against the URI of B, of the URI reference that Section 6.1 produces from r—the reference that the draft calls the equivalent URI reference (Section 2.1, Table 1). We say that B and r *disagree* when $u_1 \neq u_2$:

$$u_1(B, r) = \mathrm{conv}(\rho_{5.3}(B, r)), \qquad u_2(B, r) = \rho_{3986}\big(\mathrm{conv}(B), \mathrm{conv}(r)\big).$$

This describes two computations; it does not presuppose that the draft promises their agreement. In each case below we write the base and the reference, apply Section 5.3 step by step, apply Section 6.1 step by step, and apply RFC 3986 to the converted reference. Table 1 summarizes.

	base	CRI reference	Section 5.3 gives	Section 6.1 gives	RFC 3986 gives
A	`s://h`	`["t", null, ["p"]]`	`t://h/p`	`t:/p`	`t:/p`
B	`s://h/`	`[1]`	`s://h`	`""`	`s://h/`
B	`s://h/a/`	`[2, []]`	`s://h`	`../`	`s://h/`
C	`s://h/b/c`	`[1, "", "a"]]`	`s://h/b//a`	`/a`	`s://h/a`
B′	`s://h?q`	`[0, null, []]`	`s://h`	`""`	`s://h?q`

Table 1: The counterexamples. "Section 5.3 gives" is the resolved CRI, written as a URI; "Section 6.1 gives" is the URI reference obtained from the CRI reference; "RFC 3986 gives" is that URI reference resolved against the base.

3.1 A: a null authority is not copied

Base `["s", ["h"]]` (`s://h`); reference `["t", null, ["p"]]`, the CRI for `t:/p` (scheme `t`, authority null = NOAUTH-ROOTBASED of Figure 2, path `["p"]`).

- Step 3: a scheme is present, so discard is implicitly true. The buffer's path becomes `[]`, query empty, fragment null. The buffer's authority is `["h"]`, not true, so it is unchanged.
- Step 4: the path `["p"]` is appended: buffer path `["p"]`.
- Step 5: the non-null sections of the reference are the scheme `t` and the path. The authority is null and is not copied. Result: `["t", ["h"], ["p"]]`, i.e. `t://h/p`.

In symbols, with $B = (s, [h], [], [])$ and $r = (t, \mathit{null}, [p])$:

$$\rho_{5.3}(B, r) = (t, [h], [p]), \qquad \text{RFC 3986 for a reference with a scheme: } T = R = (t, \bot, [p]).$$

Section 2.1 assigns no authority to the equivalent URI reference of a reference whose authority is null (or not set), and RFC 3986 Section 5.2.2 gives a reference with a scheme its own authority: `t:/p`. The Section 5.3 result differs from both. The outcome does not depend on how null is read (a value, as the CDDL has it, or "not set"), since Section 2.1 covers both; nor on whether "a true authority" in step 3 refers to the buffer's or the reference's authority, since neither is true.

A second check stays inside the draft. Section 5 says that "the only requirement on a CRI reference is that reference resolution MUST yield the original CRI", and Figure 2 admits a reference with a scheme and a null authority, so a CRI may be written as a reference to itself. Applying steps 3–5 to the CRI `t:/p`, used as a reference against the base `s://h`, yields `t://h/p`, not the original; the rootless `t:p` (authority true), by contrast, is returned unchanged, because step 5 copies true.

3.2 B: a discard value without path items

Base `["s", ["h"], [""]]` (`s://h/`); reference `[1]`.

- Section 5.3: discard 1 removes the last element of the path, `[""]` becomes `[]`; query empty, fragment null; nothing to append or copy. Result `["s", ["h"], []]`, i.e. `s://h`.
- Section 6.1: discard is positive, so the path is unrooted and prefixed by $1 - 1 = 0$ copies of `../`. The rule then defines the path component only "if the CRI reference contains one or more path items"; here there are none. Read as an empty path component (the reading of our model), the URI reference is `""`. Read as undefined, Section 6.1 produces nothing and, unlike the case of discard 0 with a path, does not say that the conversion fails.
- RFC 3986, under the first reading: the empty reference has an empty path, so the target path is the base path: `s://h/`.

In symbols, with $p_B = [\varepsilon]$ and $r = (1, \bot, \bot, \bot)$:

$$\rho_{5.3}(B, r).\mathit{path} = \mathrm{take}_0([\varepsilon]) = [], \qquad \mathrm{conv}(r) = (\mathit{unrooted}, []) \text{ rendered as } \bot,$$
$$\rho_{3986}(B, \mathrm{conv}(r)).\mathit{path} = p_B = [\varepsilon].$$

Under either reading the failure clause of correction (B) is missing. The same happens for `[1, []]` and `[true]` (converted to `""`), and for `[2, []]` against `s://h/a/`: Section 5.3 removes both segments and gives `s://h`, Section 6.1 gives `../`, and RFC 3986 resolves `../` against `s://h/a/` to `s://h/`:

$$\begin{aligned}\rho_{5.3}(B, r).\mathit{path} &= \mathrm{take}_0([a, \varepsilon]) +\!\!+ [] = [],\\ \mathrm{rds}(\mathrm{merge}_B([\text{“..”}])) &= R([], [a, \text{“..”}]) = R([a], [\text{“..”}])\\ &= \mathrm{dropLast}([a]) +\!\!+ [\varepsilon] = [\varepsilon].\end{aligned}$$

A relative-path reference cannot have the meaning of `[1]` for all bases:

Lemma 1. For every base B with an authority and every non-empty segment list p, $\mathrm{last}(\mathrm{rds}(\mathrm{merge}_B(p))) = \varepsilon$ if $\mathrm{last}(p) \in \{\varepsilon, \text{“.”}, \text{“..”}\}$ and $= \mathrm{last}(p)$ otherwise; and $\rho_{3986}(B, (\cdot, \bot, \bot, \bot)).\mathit{path} = p_B$.

(In R, the last segment of the input is the last one appended, or ε in the three special cases; nothing after it is removed.) The last segment of the target therefore does not depend on B, while a reference meaning "remove the last segment" would have to end in B's second-to-last segment. Section 2.3 states that some references cannot be converted, and Section 6.1 fails for `[0, ["p"]]`; it does not fail for these.

The reference `[0, null, []]` (B′ in Table 1; "leave the path, remove the query"), which Section 2.3 lists among the references that cannot be converted, is likewise converted to `""`, which keeps the base's query under RFC 3986.

3.3 C: an empty first segment

Base `["s", ["h"], ["b", "c"]]` (`s://h/b/c`); reference `[1, ["", "a"]]`.

- Section 5.3: discard 1 removes `"c"`; the items `""` and `"a"` are appended. Result `["s", ["h"], ["b", "", "a"]]`, i.e. `s://h/b//a`.
- Section 6.1: discard 1, unrooted, no `../` prefix; the first element does not contain a colon, so no `./` prefix. The first item `""` is written without a leading slash, the second as `/a`: the path component is `"" + /a = /a`. It matches path-absolute, so the conversion succeeds with the URI reference `/a`.
- RFC 3986: a reference whose path starts with "/" replaces the base path: `s://h/a`.

In symbols, with $p_B = [b, c]$ and $r = (1, [\varepsilon, a], \bot, \bot)$:

$$\rho_{5.3}(B, r).\mathit{path} = \mathrm{take}_1([b, c]) +\!\!+ [\varepsilon, a] = [b, \varepsilon, a],$$
$$\mathrm{conv}(r) = (\mathit{unrooted}, [\varepsilon, a]) \text{ rendered as } (\mathit{rooted}, [a]), \qquad \rho_{3986}(B, \mathrm{conv}(r)).\mathit{path} = \mathrm{rds}([a]) = [a].$$

With correction (C), $\mathrm{conv}(r) = (\mathit{unrooted}, [\text{“:”}, \varepsilon, a])$ and

$$\mathrm{rds}(\mathrm{merge}_B([\text{“:”}, \varepsilon, a])) = R([], [b, \text{“:”}, \varepsilon, a]) = R([b], [\text{“:”}, \varepsilon, a]) = R([b], [\varepsilon, a]) = [b, \varepsilon, a].$$

A URI reference with the meaning of the CRI reference is `.//a`: RFC 3986 merges it to `/b/.//a` and rds gives `/b//a`. The `./` prefix that Section 6.1 prescribes for a first segment containing a colon (so that it is not read as a scheme) is also needed for an empty first segment (so that it is not read as a leading slash). The subcase `[1, [""]]`, meaning `./`, is converted to `""`.

4 Corrections

(A) Section 5.3, step 3 or step 5: “If the CRI reference has a scheme, replace the authority in the buffer with the authority of the CRI reference (which may be null or true).” Equivalently, as in RFC 3986 Section 5.2.2: “If the CRI reference has a scheme, the result is the CRI reference itself and steps 3–5 are skipped.”

(B) Section 6.1, path rule: “If the CRI reference contains a discard item (true or a positive integer) and no path items, the conversion fails.” And, for the case of Section 2.3: “If it contains a discard item of value 0, no path item, and a query item that is an empty array, the conversion fails.”

(C) Section 6.1, path rule: “If the discard value is 1 and the first element of the path is the empty string or contains a :, the path component is prefixed by ‘./’.”

5 Checks

5.1 Exhaustive enumeration

We implemented RFC 3986 Section 5.2 literally on strings (`model/rfc3986.py`); it reproduces the 42 examples of Section 5.4. We implemented Sections 5.3 and 6.1 of [1] as written (`model/cri.py`), with the corrections (A)–(C) behind a flag. For every base CRI with a host authority and every CRI reference in an enumeration (path segments from `"a"`, `"b"`, the empty string and `"x:y"`; base paths of up to three segments; references with discard 0–4 or true, or with a scheme or an authority; queries and fragments present or absent), we compared the URI of the Section 5.3 result with the RFC 3986 resolution of the Section 6.1 conversion, on the pairs that Section 6.1 converts (`model/check.py`). Where a discard value exceeds the number of base path segments, we read “remove as many elements” as stopping at the empty array (the text does not say). With $\mathcal{B}$ the 170 bases and $\mathcal{R}$ the 4,890 references, let $\mathcal{C} = \{(B, r) \in \mathcal{B} \times \mathcal{R} : \mathrm{conv}(r) \text{ defined}\}$ and $\mathcal{D} = \{(B, r) \in \mathcal{C} : u_1(B, r) \neq u_2(B, r)\}$; the corrected versions are $\mathcal{C}'$ and $\mathcal{D}'$. Then $|\mathcal{B} \times \mathcal{R}| = 831{,}300$, $|\mathcal{C}| = 720{,}120$, $|\mathcal{D}| = 93{,}810$, $|\mathcal{C}'| = 713{,}660$ and $|\mathcal{D}'| = 0$, with the classification of $\mathcal{D}$ by the shape of the reference (B comprises the ten classes discard 1–4 or true with the path absent or `[]`):

		draft -30	corrected
comparable pairs		720,120	713,660
A	scheme present, authority null	66,300	0
B	positive or true discard, no path items	10,168	0
B′	discard 0, no path, query `[]`	170	0
C	discard 1, first segment empty	17,172	0
total disagreements		93,810	0

Every disagreement is of one of the three shapes, and none remains after the corrections. The number of comparable pairs changes because correction (B) refuses 10,540 of the 720,120 pairs,

while the corrected rule (C) admits 4,080 pairs of the shape `[1, ["", "", ...]]` (converted to `.//...`) that the draft's syntax check rejects (`//...`). A smaller enumeration (paths of up to two segments, discard up to 3; 40,320 comparable pairs, 7,530 disagreements) gives the same picture.

5.2 Lean 4

`lean/Cri.lean` (Lean 4.33.1 [5], core library only, 328 lines) models paths as lists of segments, where a segment is abstracted to what the procedures inspect: `Seg = dot | dotdot | empty | other id colon` (an identifier and whether the segment contains a colon). `resolveCRI` is steps 3–5 of Section 5.3 for references in discard form; `convDraft` is the Section 6.1 path rule under the empty-path reading of Section 3, followed by the collapse that the string form imposes on an unrooted path whose first segment is empty (`"" + /a` is `/a`, `""` alone is the empty path); it omits the final path-syntax check of Section 6.1, which none of the four counterexamples triggers (the theorem concerns `convFixed`). `convFixed` applies (B) and (C); `rds`, `merge` and `resolve3986` are RFC 3986 Sections 5.2.2–5.2.4 for a base with an authority and a reference without scheme or authority.

Counterexamples by kernel evaluation. Four theorems each state that `convDraft` yields the URI reference of Table 1 and that its RFC 3986 resolution differs from `resolveCRI`:

- `cex_discard_without_path`: `[1]` against `s://h/` (B);
- `cex_discard_two_without_path`: `[2, []]` against `s://h/a/` (B);
- `cex_leading_empty_segment`: `[1, ["", "a"]]` against `s://h/b/c` (C);
- `cex_discard_zero_empty_query`: `[0, null, []]` against `s://h?q` (B′).

All are proved by `decide` and depend on `propext` only. Finding A concerns the authority, which the path model abstracts away; `authDraft` models steps 3 and 5 on the three authority values, and `cex_null_authority` states `authDraft (host 0) rootbased = host 0` with no axioms.

Theorem 1 (`convFixed_sound`)**.** *Let B be a base with an authority whose path contains no dot segments, and $r = (d, \pi, \kappa, \varphi)$ a CRI reference in discard form with π free of dot segments. Then*

$$\mathrm{conv}'(r) = u \implies \rho_{3986}(B, u) = \rho_{5.3}(B, r),$$

where conv′ *is the conversion with corrections (B) and (C) (`convFixed`).*

Proof sketch. By cases on the discard value. For true, the reference has a non-empty path p and $\mathrm{rds}(p) = p$. For 0, the reference has no path and its query is not `[]`; both sides keep the base path and take the reference's query if present, else the base's. For $k+1$, the reference has a non-empty path $s{::}t$ and the converted path is $\mathit{pre} +\!\!+ (s{::}t)$ with pre empty or [“.”] for $k = 0$ and k copies of “..” otherwise. The lemmas `rdsAux_dotdots` and `rdsAux_dot` give, for non-empty dot-free *items* and any o,

$$R(o,\ [\text{“..”}]^k +\!\!+ \mathit{items}) = \mathrm{take}_{|o|-k}(o) +\!\!+ \mathit{items}, \qquad R(o,\ \text{“.”} :: \mathit{items}) = o +\!\!+ \mathit{items}$$

(the first by induction on k, using $\mathrm{dropLast}(o) = \mathrm{take}_{|o|-1}(o)$). Since merge_B removes one more segment,

$$\mathrm{rds}(\mathrm{merge}_B(\mathit{pre} +\!\!+ (s{::}t))) = \mathrm{take}_{|p_B|-(k+1)}(p_B) +\!\!+ (s{::}t) = P_3 +\!\!+ \pi,$$

which is step 3 followed by step 4 (the case $p_B = []$, where merge_B is the identity and $\mathrm{take}_{|p_B|-(k+1)}$ is empty, is immediate). Query and fragment agree by inspection of the two definitions. □

The theorem depends on `propext` and `Quot.sound`; `Classical.choice` is not used and the file contains no `sorry`. `lean Cri.lean` prints only the six `#print axioms` lines. The theorem covers references in discard form; correction (A) concerns the authority, which this model abstracts away, and rests on the enumeration.

Relation between the two checks. The segment-level rendering of RFC 3986 in Lean was compared with the literal string implementation on all 1,364 rooted paths of one to five segments over {`a`, empty, `.`, `..`} and on all 21,420 (draft) and 19,320 (corrected) convertible pairs of a host base and a discard-form reference of the smaller enumeration, with no difference (`model/seglevel.py`); `lean/CrossCheck.lean` re-checks by `decide` the results that the Python model computed for 60 random pairs.

6 A remark on host names

Section 3 of [1] states the objective that two distinct normalized CRIs do not convert to the same URI. Constraint C5 allows the host name `["1","2","1","2"]`, which converts to `1.2.1.2`, as does the IP-address host `h'01020102'`; RFC 3986 Section 3.2.2 says that a host matching IPv4address is an IPv4 address literal, and Section 8.1.2 of [1] maps such a Uri-Host to host-ip. A sentence in C5 ("A registered name that matches the IPv4address rule MUST be represented as host-ip") would make the data model say the same.

7 Scope

We claim exactly the following. (1) On the inputs of Table 1, Sections 5.3 and 6.1 of draft-ietf-core-href-30, applied as written, give the results shown, which differ from the authority that Section 2.1 and RFC 3986 give (A) or from the RFC 3986 meaning of the converted reference (B, C, B′). (2) With corrections (A)–(C), no such disagreement exists on the enumeration of Section 5, and the Lean theorem extends (B) and (C) to all path lengths for bases with an authority and discard-form references, at the segment level. We do not claim anything about base CRIs without an authority (there RFC 3986's own rds turns rootless paths into rooted ones, so we leave them aside), about the practical frequency of the shapes involved, or about consequences beyond the plain fact that a converted reference denotes a different URI than the CRI reference. The segment-level model is related to the string procedures by testing, not by proof. The reference shape `[null, true, path]` was excluded from the enumeration.

In the 144 closed issues and pull requests of `core-wg/href` and the messages of the `core` mailing list that we examined (listed in `prior_art.md`) (2020 onwards, about 90 matching *href*, *CRI*, *remove_dot_segments*, *rootless* or *discard*; twelve read in full), none of A, B or C is discussed; issue #48 introduced the `./` prefix for a leading colon, and issue #30 found the wording of step 5 odd without drawing consequence A.

Artifacts

`lean/Cri.lean`, `lean/CrossCheck.lean` (Lean 4.33.1, core only) and `model/` (Python 3: `rfc3986.py`, `cri.py`, `check.py`, `inject.py`, `seglevel.py`, `gen_crosscheck.py`, with their outputs, produced by re-running the scripts in the archive), and `prior_art.md` (the issues and messages examined). Checked against the text of draft-ietf-core-href-30 (21 November 2025). Repository and archive: to be published with this note.

Acknowledgements

An AI agent, applying a target-selection method set by the author, chose this draft and produced the proofs, code and text. The author restricted the claims to those with self-evident evidence, directed two rounds of revision, and checked and approved the results.

References

[1] C. Bormann (Ed.) and H. Birkholz, "Constrained Resource Identifiers," Internet-Draft draft-ietf-core-href-30, IETF, 21 November 2025; in the RFC Editor queue as of October 2026.

[2] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," STD 66, RFC 3986, January 2005.

[3] C. Bormann and P. Hoffman, "Concise Binary Object Representation (CBOR)," STD 94, RFC 8949, December 2020.

[4] H. Birkholz, C. Vigano, and C. Bormann, "Concise Data Definition Language (CDDL)," RFC 8610, June 2019.

[5] L. de Moura and S. Ullrich, "The Lean 4 Theorem Prover and Programming Language," in *CADE 28*, LNCS 12699, Springer, 2021.