A Formalization of Web Components

Achim D. Brucker Michael Herzberg

March 17, 2025

*Department of Computer Science, University of Exeter, Exeter, UK
a.brucker@exeter.ac.uk

† Department of Computer Science, The University of Sheffield, Sheffield, UK
msherzberg1@sheffield.ac.uk

Abstract

While the DOM with shadow trees provide the technical basis for defining web components, the DOM standard neither defines the concept of web components nor specifies the safety properties that web components should guarantee. Consequently, the standard also does not discuss how or even if the methods for modifying the DOM respect component boundaries.

In AFP entry, we present a formally verified model of web components and define safety properties which ensure that different web components can only interact with each other using well-defined interfaces. Moreover, our verification of the application programming interface (API) of the DOM revealed numerous invariants that implementations of the DOM API need to preserve to ensure the integrity of components.

Keywords: Web Components, DOM

Contents

1 Introduction

The trend towards ever more complex client-side web applications is unstoppable. Compared to traditional software development, client-side web development lacks a well-established component model which allows easily and safely reusing implementations. The Document Object Model (DOM) essentially defines a tree-like data structure (the *node tree*) for representing documents in general and HTML documents in particular.

Shadow trees are a recent addition to the DOM standard [7] to enable web developers to partition the node tree into "sub-trees." The vision of shadow trees is to enable web developers to provide a library of re-usable and customizable widgets. For example, let us consider a multi-tab view called *Fancy Tab*, which is a simplified version of [1].

The left-hand side of Figure 1.1 shows the rendered output of the widget in use while the right-hand side shows the HTML source code snippet. It provides a custom HTML tag `<fancy-tabs>` using an HTML template that developers can use to include the widget. Its children will be rendered inside the widget, more precisely, inside its *slots* (elements of type `slot`). It has a slot called "title" and a default slot, which receives all children that do not specify a "slot" attribute.

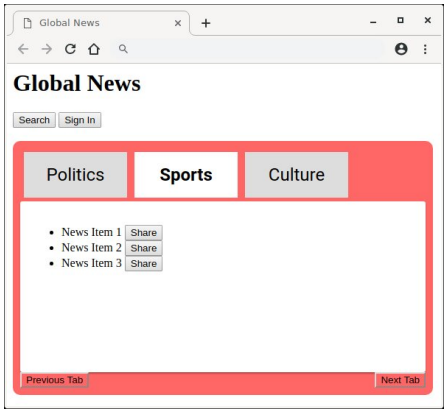
It is important to understand that slotting does *not change* the structure of the DOM (i. e., the underlying pointer graph): instead, slotting is implemented using special element attributes such as "slot," which control the final rendering. The DOM standard specifies methods that inspect the effect of these attributes such as `assigned_slot`, but the majority of DOM methods do not consider the semantics of these attributes and therefore do not traverse into shadow trees.

This provides an important boundary for client-side code. For example, a JavaScript program coming from the widget developer that changes the style attributes of the "Previous Tab" and "Next Tab" buttons in the lower corners of the widget will not affect buttons belonging to other parts coming from outside, i. e., the application of the widget consumer. Similarly, a JavaScript program that changes the styles of buttons outside of Fancy Tab, such as the navigation buttons, will not have any effect on them, even in the case of duplicate identifiers.

Sadly, the DOM standard neither defines the concept of web components nor specifies the safety properties that they should guarantee, not even informally. Consequently, the standard also does not discuss how or even if the methods for modifying the node tree respect component boundaries. Thus, shadow roots are only the very first step in defining a safe web component model.

Earlier [2, 3], we presented a formalization of the "flat" DOM (called Core DOM) without any support for shadow trees or components. We then extended this formalisation with support for shadow trees and slots [4].

In this AFP entries, we use the basis provided by our earlier work for defining a *formally verified model of web components* in general and, in particular, the notion of *weak* and *strong component safety*. For all methods that query, modify, or transform the DOM, we formally analyze their level of component safety. In more detail,



(a) User view

```
<fancy-tabs>
  <button slot="title">Politics</button>
  <button slot="title" selected>Sports</button>
  <button slot="title">Culture</button>
  <section>content panel 1</section>
  <ul>
    <li>News Item 1 <button>Share</button></li>
    <li>News Item 2 <button>Share</button></li>
    <li>News Item 3 <button>Share</button></li>
  </ul>
  <section>content panel 3</section>
</fancy-tabs>
```

(b) Consumer view

Figure 1.1: A simple example: a fancy tab component.

the contribution of this AFP entry is four-fold:

1. We provide a formal model of web components and their safety guarantees to web developers, enabling a compositional development of web applications,
2. for each method, we formally verify that it is either weakly or strongly component safe, or we provide a proof showing that it is not component safe,
3. we fill the gaps in the standard by explicitly formalizing invariants that are left out in the standard. These invariants are required to ensure that methods in the standard preserve a valid node tree. Finally,
4. we present a formal model of the DOM with shadow roots including the methods for querying, modifying, and transforming DOM instances with shadow roots.

Overall, our work gives web developers the guarantee that their code will respect the component boundaries as long as they abstain from or are careful when using certain DOM methods such as `appendChild` or `ownerDocument`.

The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle (we refer readers interested in a more high-level presentation of the work to [5, 6]. The structure follows the theory dependencies (see Figure 1.2).

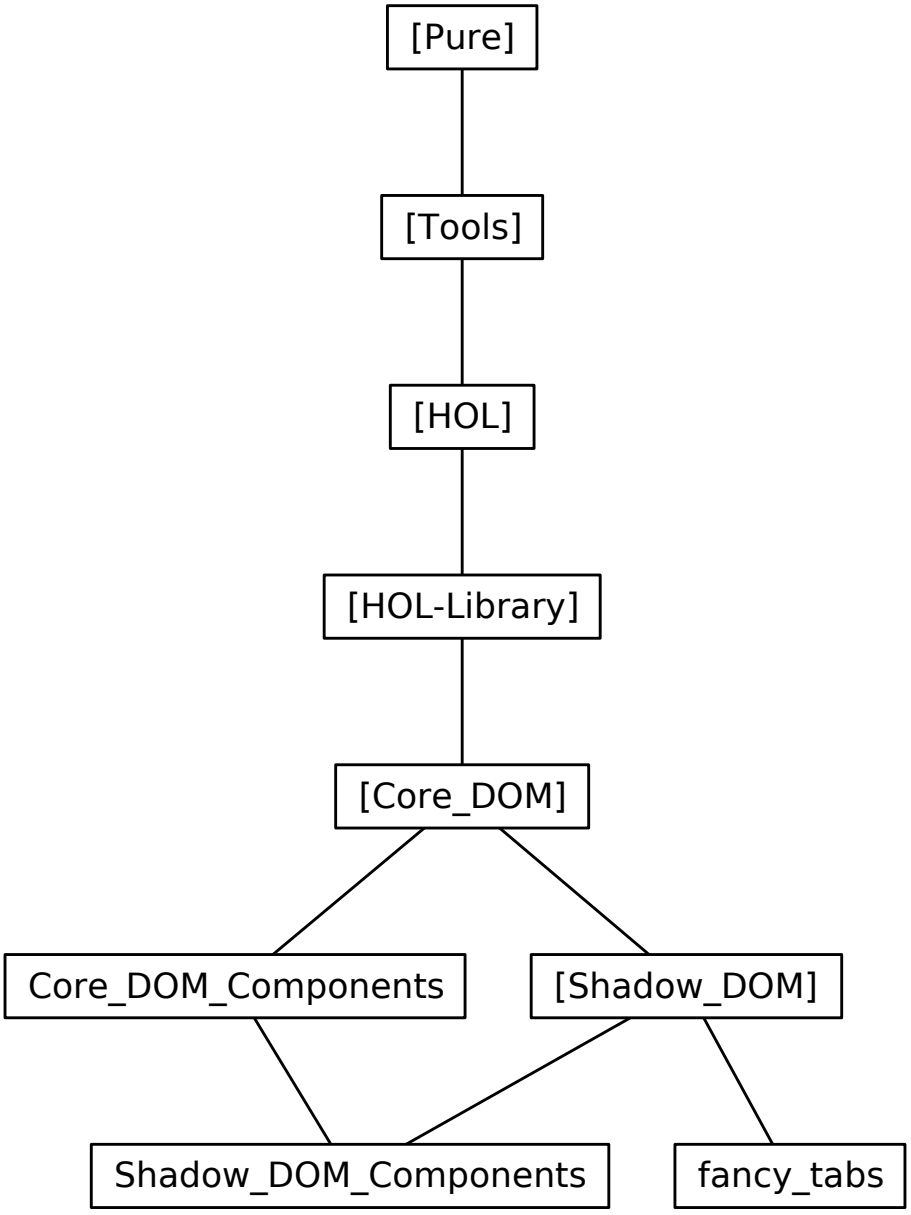


Figure 1.2: The Dependency Graph of the Isabelle Theories.

2 Web Components

2.1 DOM Components (Core_DOM_Components)

```
theory Core_DOM_Components
  imports Core_DOM.Core_DOM
begin


locale l_get_component_Core_DOM_defs =
  l_get_root_node_defs get_root_node get_root_node_locs +
  l_to_tree_order_defs to_tree_order
  for get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
begin

definition a_get_component :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  where
    "a_get_component ptr = do {
      root ← get_root_node ptr;
      to_tree_order root
    }"

definition a_is_strongly_dom_component_safe ::
  "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  where
    "a_is_strongly_dom_component_safe S_arg S_result h h' = (
      let removed_pointers = fset (object_ptr_kinds h) - fset (object_ptr_kinds h') in
      let added_pointers = fset (object_ptr_kinds h') - fset (object_ptr_kinds h) in
      let arg_components =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_component ptr|_r) ∩ fset (object_ptr_kinds h).
        set |h ⊢ a_get_component ptr|_r) in
      let arg_components' =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_component ptr|_r) ∩ fset (object_ptr_kinds h').
        set |h' ⊢ a_get_component ptr|_r) in
      removed_pointers ⊆ arg_components ∧
      added_pointers ⊆ arg_components' ∧
      S_result ⊆ arg_components' ∧
      (∀ outside_ptr ∈ fset (object_ptr_kinds h) ∩ fset (object_ptr_kinds h') -
        (⋃ptr ∈ S_arg. set |h  ⊢ a_get_component ptr|_r). preserved (get_M outside_ptr id) h h'))"

definition a_is_weakly_dom_component_safe ::
  "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  where
    "a_is_weakly_dom_component_safe S_arg S_result h h' = (
      let removed_pointers = fset (object_ptr_kinds h) - fset (object_ptr_kinds h') in
      let added_pointers = fset (object_ptr_kinds h') - fset (object_ptr_kinds h) in
      let arg_components =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_component ptr|_r) ∩ fset (object_ptr_kinds h).
        set |h ⊢ a_get_component ptr|_r) in
      let arg_components' =
        (⋃ptr ∈ (⋃ptr ∈ S_arg. set |h ⊢ a_get_component ptr|_r) ∩ fset (object_ptr_kinds h').
        set |h' ⊢ a_get_component ptr|_r) in
      removed_pointers ⊆ arg_components ∧
      S_result ⊆ arg_components' ∪ added_pointers  ∧
      (∀ outside_ptr ∈ fset (object_ptr_kinds h) ∩ fset (object_ptr_kinds h') -
```

```
(⋃ptr ∈ S_arg. set |h ⊢ a_get_component ptr|_r). preserved (get_M outside_ptr id) h h'))"

lemma "a_is_strongly_dom_component_safe S_arg S_result h h' ⟹ a_is_weakly_dom_component_safe S_arg S_result
h h'"
  by(auto simp add: a_is_strongly_dom_component_safe_def a_is_weakly_dom_component_safe_def Let_def)

definition is_document_component :: "(_) object_ptr list ⇒ bool"
  where
    "is_document_component c = is_document_ptr_kind (hd c)"

definition is_disconnected_component :: "(_) object_ptr list ⇒ bool"
  where
    "is_disconnected_component c = is_node_ptr_kind (hd c)"
end

global_interpretation l_get_component_Core_DOM_defs get_root_node get_root_node_locs to_tree_order
  defines get_component = a_get_component
    and is_strongly_dom_component_safe = a_is_strongly_dom_component_safe
    and is_weakly_dom_component_safe = a_is_weakly_dom_component_safe
  .


locale l_get_component_defs =
  fixes get_component :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  fixes is_strongly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
  fixes is_weakly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"

locale l_get_component_Core_DOM =
  l_to_tree_order_wf +
  l_get_component_defs +
  l_get_component_Core_DOM_defs +
  l_get_ancestors +
  l_get_ancestors_wf +
  l_get_root_node +
  l_get_root_node_wf_Core_DOM +
  l_get_parent +
  l_get_parent_wf +
  l_get_element_by +
  l_to_tree_order_wf_get_root_node_wf +

  assumes get_component_impl: "get_component = a_get_component"
  assumes is_strongly_dom_component_safe_impl:
    "is_strongly_dom_component_safe = a_is_strongly_dom_component_safe"
  assumes is_weakly_dom_component_safe_impl:
    "is_weakly_dom_component_safe = a_is_weakly_dom_component_safe"
begin
lemmas get_component_def = a_get_component_def[folded get_component_impl]
lemmas is_strongly_dom_component_safe_def =
  a_is_strongly_dom_component_safe_def[folded is_strongly_dom_component_safe_impl]
lemmas is_weakly_dom_component_safe_def =
  a_is_weakly_dom_component_safe_def[folded is_weakly_dom_component_safe_impl]

lemma get_dom_component_ptr_in_heap:
  assumes "h ⊢ ok (get_component ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  using assms get_root_node_ptr_in_heap
  by(auto simp add: get_component_def)

lemma get_component_ok:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "ptr |∈| object_ptr_kinds h"
```

```
  shows "h ⊢ ok (get_component ptr)"
  using assms
  apply(auto simp add: get_component_def a_get_root_node_def intro!: bind_is_OK_pure_I)[1]
  using get_root_node_ok  to_tree_order_ok get_root_node_ptr_in_heap
   apply blast
  by (simp add: local.get_root_node_root_in_heap local.to_tree_order_ok)

lemma get_dom_component_ptr:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  shows "ptr ∈ set c"
proof(insert assms(1) assms(4), induct ptr rule: heap_wellformed_induct_rev )
  case (step child)
  then show ?case
  proof (cases "is_node_ptr_kind child")
    case True
    obtain node_ptr where
      node_ptr: "cast_{node_ptr2object_ptr} node_ptr = child"
      using ‹is_node_ptr_kind child› node_ptr_casts_commute3 by blast
    have "child |∈| object_ptr_kinds h"
      using ‹h ⊢ get_component child →r c› get_dom_component_ptr_in_heap by fast
    with node_ptr have "node_ptr |∈| node_ptr_kinds h"
      by auto
    then obtain parent_opt where
      parent: "h ⊢ get_parent node_ptr →r parent_opt"
      using get_parent_ok ‹type_wf h› ‹known_ptrs h›
      by fast
    then show ?thesis
    proof (induct parent_opt)
      case None
      then have "h ⊢ get_root_node (cast node_ptr) →r cast node_ptr"
        by (simp add: local.get_root_node_no_parent)
      then show ?case
        using ‹type_wf h› ‹known_ptrs h› node_ptr step(2)
        apply(auto simp add: get_component_def a_get_root_node_def elim!: bind_returns_result_E2)[1]
        using to_tree_order_ptr_in_result returns_result_eq by fastforce
    next
      case (Some parent_ptr)
      then have "h ⊢ get_component parent_ptr →r c"
        using step(2) node_ptr ‹type_wf h› ‹known_ptrs h› ‹heap_is_wellformed h›
          get_root_node_parent_same
        by(auto simp add: get_component_def elim!: bind_returns_result_E2 intro!: bind_pure_returns_result_I)
      then have "parent_ptr ∈ set c"
        using step node_ptr Some by blast
      then show ?case
        using  ‹type_wf h› ‹known_ptrs h› ‹heap_is_wellformed h› step(2) node_ptr Some
        apply(auto simp add: get_component_def elim!: bind_returns_result_E2)[1]
        using to_tree_order_parent by blast
    qed
  next
    case False
    then show ?thesis
      using ‹type_wf h› ‹known_ptrs h› step(2)
      apply(auto simp add: get_component_def elim!: bind_returns_result_E2)[1]
      by (metis is_OK_returns_result_I local.get_root_node_not_node_same
          local.get_root_node_ptr_in_heap local.to_tree_order_ptr_in_result returns_result_eq)
  qed
qed

lemma get_component_parent_inside:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "cast node_ptr ∈ set c"
```

```
  assumes "h ⊢ get_parent node_ptr →r Some parent"
  shows "parent ∈ set c"
proof -
  have "parent |∈| object_ptr_kinds h"
    using assms(6) get_parent_parent_in_heap by blast

  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr" and
    c: "h ⊢ to_tree_order root_ptr →r c"
    using assms(4)
    by (metis (no_types, opaque_lifting) bind_returns_result_E2 get_component_def get_root_node_pure)
  then have "h ⊢ get_root_node (cast node_ptr) →r root_ptr"
    using assms(1) assms(2) assms(3) assms(5) to_tree_order_same_root by blast
  then have "h ⊢ get_root_node parent →r root_ptr"
    using assms(6)  get_root_node_parent_same by blast
  then have "h ⊢ get_component parent →r c"
    using c get_component_def by auto
  then show ?thesis
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
qed

lemma get_component_subset:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "ptr' ∈ set c"
  shows "h ⊢ get_component ptr' →r c"
proof(insert assms(1) assms(5), induct ptr' rule: heap_wellformed_induct_rev )
  case (step child)
  then show ?case
  proof (cases "is_node_ptr_kind child")
    case True
    obtain node_ptr where
      node_ptr: "cast_node_ptr2object_ptr node_ptr = child"
      using ‹is_node_ptr_kind child› node_ptr_casts_commute3 by blast
    have "child |∈| object_ptr_kinds h"
      using to_tree_order_ptrs_in_heap assms(1) assms(2) assms(3) assms(4) step(2)
      unfolding get_component_def
      by (meson bind_returns_result_E2 get_root_node_pure)
    with node_ptr have "node_ptr |∈| node_ptr_kinds h"
      by auto
    then obtain parent_opt where
      parent: "h ⊢ get_parent node_ptr →r parent_opt"
      using get_parent_ok ‹type_wf h› ‹known_ptrs h›
      by fast
    then show ?thesis
    proof (induct parent_opt)
      case None
      then have "h ⊢ get_root_node child →r child"
        using assms(1) get_root_node_no_parent node_ptr by blast
      then show ?case
        using ‹type_wf h› ‹known_ptrs h› node_ptr step(2) assms(4) assms(1)
        by (metis (no_types) bind_pure_returns_result_I2 bind_returns_result_E2 get_component_def
            get_root_node_pure is_OK_returns_result_I returns_result_eq to_tree_order_same_root)
    next
      case (Some parent_ptr)
      then have "h ⊢ get_component parent_ptr →r c"
        using step get_component_parent_inside assms node_ptr by blast
      then show ?case
        using Some node_ptr
        apply(auto simp add: get_component_def elim!: bind_returns_result_E2
            del: bind_pure_returns_result_I intro!: bind_pure_returns_result_I)[1]
        using get_root_node_parent_same by blast
    qed
  next
```

```
    case False
    then have "child |∈| object_ptr_kinds h"
      using assms(1) assms(4) step(2)
      by (metis (no_types, lifting) assms(2) assms(3) bind_returns_result_E2 get_root_node_pure
          get_component_def to_tree_order_ptrs_in_heap)
    then have "h ⊢ get_root_node child →r child"
      using assms(1) False get_root_node_not_node_same by blast
    then show ?thesis
      using assms(1) assms(2) assms(3) assms(4) step.prems
      by (metis (no_types) False ‹child |∈| object_ptr_kinds h› bind_pure_returns_result_I2
          bind_returns_result_E2 get_component_def get_root_node_ok get_root_node_pure returns_result_eq
          to_tree_order_node_ptrs)
  qed
qed

lemma get_component_to_tree_order_subset:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ to_tree_order ptr →r nodes"
  assumes "h ⊢ get_component ptr →r c"
  shows "set nodes ⊆ set c"
  using assms
  apply(auto simp add: get_component_def elim!: bind_returns_result_E2)[1]
  by (meson to_tree_order_subset  assms(5) contra_subsetD get_dom_component_ptr)

lemma get_component_to_tree_order:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ to_tree_order ptr' →r to"
  assumes "ptr ∈ set to"
  shows "h ⊢ get_component ptr' →r c"
  by (metis (no_types, opaque_lifting) assms(1) assms(2) assms(3) assms(4) assms(5) assms(6)
      get_component_ok get_component_subset get_component_to_tree_order_subset is_OK_returns_result_E
      local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap select_result_I2 subsetCE)

lemma get_component_root_node_same:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ get_root_node ptr →r root_ptr"
  assumes "x ∈ set c"
  shows "h ⊢ get_root_node x →r root_ptr"
proof(insert assms(1) assms(6), induct x rule: heap_wellformed_induct_rev )
  case (step child)
  then show ?case
  proof (cases "is_node_ptr_kind child")
    case True
    obtain node_ptr where
      node_ptr: "cast_node_ptr2object_ptr node_ptr = child"
      using ‹is_node_ptr_kind child› node_ptr_casts_commute3 by blast
    have "child |∈| object_ptr_kinds h"
      using to_tree_order_ptrs_in_heap assms(1) assms(2) assms(3) assms(4) step(2)
      unfolding get_component_def
      by (meson bind_returns_result_E2 get_root_node_pure)
    with node_ptr have "node_ptr |∈| node_ptr_kinds h"
      by auto
    then obtain parent_opt where
      parent: "h ⊢ get_parent node_ptr →r parent_opt"
      using get_parent_ok ‹type_wf h› ‹known_ptrs h›
      by fast
    then show ?thesis
    proof (induct parent_opt)
      case None
      then have "h ⊢ get_root_node child →r child"
        using assms(1) get_root_node_no_parent  node_ptr by blast
```

```
      then show ?case
        using ‹type_wf h› ‹known_ptrs h› node_ptr step(2) assms(4) assms(1) assms(5)
        by (metis (no_types) ‹child |∈| object_ptr_kinds h› bind_pure_returns_result_I
            get_component_def get_dom_component_ptr get_component_subset get_root_node_pure is_OK_returns_result_E
            returns_result_eq to_tree_order_ok to_tree_order_same_root)
    next
      case (Some parent_ptr)
      then have "h ⊢ get_component parent_ptr →r c"
        using step get_component_parent_inside assms node_ptr
        by (meson get_component_subset)
      then show ?case
        using Some node_ptr
        apply(auto simp add: get_component_def elim!: bind_returns_result_E2)[1]
        using get_root_node_parent_same
        using ‹h ⊢ get_component parent_ptr →r c› assms(1) assms(2) assms(3) get_dom_component_ptr
          step.hyps
        by blast
    qed
  next
    case False
    then have "child |∈| object_ptr_kinds h"
      using assms(1) assms(4) step(2)
      by (metis (no_types, lifting) assms(2) assms(3) bind_returns_result_E2 get_root_node_pure
          get_component_def to_tree_order_ptrs_in_heap)
    then have "h ⊢ get_root_node child →r child"
      using assms(1) False get_root_node_not_node_same by auto
    then show ?thesis
      using assms(1) assms(2) assms(3) assms(4) step.prems assms(5)
      by (metis (no_types, opaque_lifting) bind_returns_result_E2 get_component_def get_root_node_pure
          returns_result_eq to_tree_order_same_root)
  qed
qed


lemma get_dom_component_no_overlap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ get_component ptr' →r c'"
  shows "set c ∩ set c' = {} ∨ c = c'"
proof (rule ccontr, auto)
  fix x
  assume 1: "c ≠ c'" and 2: "x ∈ set c" and 3: "x ∈ set c'"
  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    using assms(4) unfolding get_component_def
    by (meson bind_is_OK_E is_OK_returns_result_I)
  moreover obtain root_ptr' where root_ptr': "h ⊢ get_root_node ptr' →r root_ptr'"
    using assms(5) unfolding get_component_def
    by (meson bind_is_OK_E is_OK_returns_result_I)
  ultimately have "root_ptr ≠ root_ptr'"
    using 1 assms
    unfolding get_component_def
    by (meson bind_returns_result_E3 get_root_node_pure returns_result_eq)

  moreover have "h ⊢ get_root_node x →r root_ptr"
    using 2 root_ptr get_component_root_node_same assms by blast
  moreover have "h ⊢ get_root_node x →r root_ptr'"
    using 3 root_ptr' get_component_root_node_same assms by blast
  ultimately show False
    using select_result_I2 by force
qed

lemma get_component_separates_tree_order:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
```

```
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ to_tree_order ptr →r to"
  assumes "h ⊢ get_component ptr' →r c'"
  assumes "ptr' ∉ set c"
  shows "set to ∩ set c' = {}"
proof -
  have "c ≠ c'"
    using assms get_dom_component_ptr by blast
  then have "set c ∩ set c' = {}"
    using assms get_dom_component_no_overlap by blast
  moreover have "set to ⊆ set c"
    using assms get_component_to_tree_order_subset by blast
  ultimately show ?thesis
    by blast
qed

lemma get_component_separates_tree_order_general:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ to_tree_order ptr'' →r to''"
  assumes "ptr'' ∈ set c"
  assumes "h ⊢ get_component ptr' →r c'"
  assumes "ptr' ∉ set c"
  shows "set to'' ∩ set c' = {}"
proof -
  obtain root_ptr where root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    using assms(4)
    by (metis bind_is_OK_E get_component_def is_OK_returns_result_I)
  then have c: "h ⊢ to_tree_order root_ptr →r c"
    using assms(4) unfolding get_component_def
    by (simp add: bind_returns_result_E3)
  with root_ptr show ?thesis
    using assms get_component_separates_tree_order get_component_subset
    by meson
qed
end

interpretation i_get_component?: l_get_component_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name
  by(auto simp add: l_get_component_Core_DOM_def l_get_component_Core_DOM_axioms_def get_component_def
      is_strongly_dom_component_safe_def is_weakly_dom_component_safe_def instances)
declare l_get_component_Core_DOM_axioms [instances]
```

2.1.1 get_child_nodes

```
locale l_get_component_get_child_nodes_Core_DOM =
  l_get_component_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_dom_component_get_child_nodes:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ get_child_nodes ptr' →r children"
  assumes "child ∈ set children"
  shows "cast child ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "cast child ∈ set c"
  obtain root_ptr where
```

```
    root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    by (metis assms(4) bind_is_OK_E get_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node (cast child) →r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →r root_ptr"
    using assms(1) assms(2) assms(3) assms(5) assms(6) local.child_parent_dual
      local.get_root_node_parent_same
    by blast
  then have "h ⊢ get_component ptr' →r c"
    using "1" assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) local.child_parent_dual
      local.get_component_parent_inside local.get_component_subset
    by blast
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr
    by blast
next
  assume 1: "ptr' ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    by (metis assms(4) bind_is_OK_E get_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node ptr' →r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node (cast child) →r root_ptr"
    using assms(1) assms(2) assms(3) assms(5) assms(6) local.child_parent_dual local.get_root_node_parent_same
    by blast
  then have "h ⊢ get_component ptr' →r c"
    using "1" assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) local.child_parent_dual
      local.get_component_parent_inside local.get_component_subset
    by blast
  then show "cast_node_ptr2object_ptr child ∈ set c"
    by (smt ‹h ⊢ get_root_node (cast_node_ptr2object_ptr child) →r root_ptr› assms(1) assms(2) assms(3)
        assms(5) assms(6) disjoint_iff_not_equal is_OK_returns_result_E is_OK_returns_result_I
        l_get_component_Core_DOM.get_dom_component_no_overlap local.child_parent_dual local.get_component_ok
        local.get_component_parent_inside local.get_dom_component_ptr local.get_root_node_ptr_in_heap
        local.l_get_component_Core_DOM_axioms)
qed


lemma get_child_nodes_get_component:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ get_child_nodes ptr →r children"
  shows "cast ` set children ⊆ set c"
  using assms get_dom_component_get_child_nodes
  using local.get_dom_component_ptr by auto


lemma get_child_nodes_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_child_nodes ptr →r children"
  assumes "h ⊢ get_child_nodes ptr →h h'"
  shows "is_strongly_dom_component_safe {ptr} (cast ` set children) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_child_nodes_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def)[1]
    by (smt IntI get_dom_component_get_child_nodes is_OK_returns_result_E
        is_OK_returns_result_I local.get_child_nodes_ptr_in_heap local.get_component_impl
```

```
        local.get_component_ok local.get_dom_component_ptr select_result_I2)
qed
end

interpretation i_get_component_get_child_nodes?: l_get_component_get_child_nodes_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_component_get_child_nodes_Core_DOM_def instances)
declare l_get_component_get_child_nodes_Core_DOM_axioms [instances]
```

2.1.2 get_parent

```
locale l_get_component_get_parent_Core_DOM =
  l_get_component_Core_DOM +
  l_get_parent_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_parent_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →_r c"
  assumes "h ⊢ get_parent ptr' →_r Some parent"
  shows "parent ∈ set c ⟷ cast ptr' ∈ set c"
  by (meson assms(1) assms(2) assms(3) assms(4) assms(5) is_OK_returns_result_E
      l_to_tree_order_wf.to_tree_order_parent local.get_component_parent_inside local.get_component_subset
      local.get_component_to_tree_order_subset local.get_parent_parent_in_heap local.l_to_tree_order_wf_axioms
      local.to_tree_order_ok local.to_tree_order_ptr_in_result subsetCE)


lemma get_parent_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_parent node_ptr →_r Some parent"
  assumes "h ⊢ get_parent node_ptr →_h h'"
  shows "is_strongly_dom_component_safe {cast node_ptr} {parent} h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_parent_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def)[1]
    using get_dom_component_get_child_nodes local.get_component_impl local.get_dom_component_ptr
    by (metis (no_types, opaque_lifting) Int_iff get_parent_is_strongly_dom_component_safe_step
        local.get_component_ok local.get_parent_parent_in_heap local.to_tree_order_ok local.to_tree_order_parent
        local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap returns_result_select_result)
qed
end

interpretation i_get_component_get_parent?: l_get_component_get_parent_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_component_get_parent_Core_DOM_def instances)
declare l_get_component_get_parent_Core_DOM_axioms [instances]
```

2.1.3 get_root_node

```
locale l_get_component_get_root_node_Core_DOM =
```

```
  l_get_component_Core_DOM +
  l_get_root_node_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_root_node_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →_r c"
  assumes "h ⊢ get_root_node ptr' →_r root"
  shows "root ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "root ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →_r root_ptr"
    by (metis assms(4) bind_is_OK_E get_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node root →_r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_component_root_node_same root_ptr
    by blast
  moreover have "h ⊢ get_root_node ptr' →_r root_ptr"
    by (metis (no_types, lifting) calculation assms(1) assms(2) assms(3) assms(5)
        is_OK_returns_result_E local.get_root_node_root_in_heap local.to_tree_order_ok
        local.to_tree_order_ptr_in_result local.to_tree_order_same_root select_result_I2)
  ultimately have "h ⊢ get_component ptr' →_r c"
    apply(auto simp add: get_component_def)[1]
    by (smt "1" assms(1) assms(2) assms(3) assms(4) bind_pure_returns_result_I bind_returns_result_E3
        local.get_component_def local.get_component_subset local.get_root_node_pure)
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume 1: "ptr' ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →_r root_ptr"
    by (metis assms(4) bind_is_OK_E get_component_def is_OK_returns_result_I)
  have "h ⊢ get_root_node ptr' →_r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node root →_r root_ptr"
    by (metis assms(1) assms(2) assms(3) assms(5) is_OK_returns_result_E local.get_root_node_root_in_heap
        local.to_tree_order_ok local.to_tree_order_ptr_in_result local.to_tree_order_same_root returns_result_eq)
  then have "h ⊢ get_component ptr' →_r c"
    using "1" assms(1) assms(2) assms(3) assms(4) local.get_component_subset by blast
  then show "root ∈ set c"
    using assms(5) bind_returns_result_E3 local.get_component_def local.to_tree_order_ptr_in_result
    by fastforce
qed

lemma get_root_node_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node ptr →_r root"
  assumes "h ⊢ get_root_node ptr →_h h'"
  shows "is_strongly_dom_component_safe {ptr} {root} h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson local.get_root_node_pure pure_returns_heap_eq)
  then show ?thesis
    using assms
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def)[1]
    by (metis (no_types, opaque_lifting) IntI get_root_node_is_strongly_dom_component_safe_step is_OK_returns_resu
        local.get_component_impl local.get_component_ok local.get_dom_component_ptr
        local.get_root_node_ptr_in_heap returns_result_select_result)
qed
end
```

```
interpretation i_get_component_get_root_node?: l_get_component_get_root_node_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_component_get_root_node_Core_DOM_def instances)
declare l_get_component_get_root_node_Core_DOM_axioms [instances]
```

2.1.4 get_element_by_id

```
locale l_get_component_get_element_by_id_Core_DOM =
  l_get_component_Core_DOM +
  l_first_in_tree_order_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_element_by_id_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →_r c"
  assumes "h ⊢ get_element_by_id ptr' idd →_r Some result"
  shows "cast result ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "cast result ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →_r root_ptr"
    by (metis assms(4) bind_is_OK_E get_component_def is_OK_returns_result_I)
  then have "h ⊢ to_tree_order root_ptr →_r c"
    using ‹h ⊢ get_component ptr →_r c›
    by (simp add: get_component_def bind_returns_result_E3)

  obtain to' where to': "h ⊢ to_tree_order ptr' →_r to'"
    using ‹h ⊢ get_element_by_id ptr' idd →_r Some result›
    apply(simp add: get_element_by_id_def first_in_tree_order_def)
    by (meson bind_is_OK_E is_OK_returns_result_I)
  then have "cast result ∈ set to'"
    using ‹h ⊢ get_element_by_id ptr' idd →_r Some result› get_element_by_id_result_in_tree_order
    by blast

  have "h ⊢ get_root_node (cast result) →_r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →_r root_ptr"
    using ‹cast result ∈ set to'› ‹h ⊢ to_tree_order ptr' →_r to'›
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_ptr get_component_root_node_same
      get_component_subset get_component_to_tree_order
    by blast
  then have "h ⊢ get_component ptr' →_r c"
    using ‹h ⊢ to_tree_order root_ptr →_r c›
    using get_component_def by auto
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume "ptr' ∈ set c"
  moreover obtain to' where to': "h ⊢ to_tree_order ptr' →_r to'"
    by (meson assms(1) assms(2) assms(3) assms(4) calculation get_dom_component_ptr_in_heap
        get_component_subset is_OK_returns_result_E is_OK_returns_result_I to_tree_order_ok)
  ultimately have "set to' ⊆ set c"
    using assms(1) assms(2) assms(3) assms(4) get_component_subset get_component_to_tree_order_subset
    by blast
  moreover have "cast result ∈ set to'"
```

```
    using assms(5) get_element_by_id_result_in_tree_order to' by blast
  ultimately show "cast result ∈ set c"
    by blast
qed


lemma get_element_by_id_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_element_by_id ptr idd →r Some result"
  assumes "h ⊢ get_element_by_id ptr idd →h h'"
  shows "is_strongly_dom_component_safe {ptr} {cast result} h h'"
proof -
  have "h = h'"
    using assms(5)
    by(auto simp add: preserved_def get_element_by_id_def first_in_tree_order_def
        elim!: bind_returns_heap_E2 intro!: map_filter_M_pure bind_pure_I split: option.splits list.splits)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_element_by_id_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast result ∈ set to"
    using assms(4) local.get_element_by_id_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_component ptr →r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_component_impl
      local.get_component_ok
    by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def get_element_by_id_def
        first_in_tree_order_def elim!: bind_returns_result_E2 intro!: map_filter_M_pure bind_pure_I
        split: option.splits list.splits)[1]
    by (metis (no_types, opaque_lifting) Int_iff ‹ptr |∈| object_ptr_kinds h› assms(4)
        get_element_by_id_is_strongly_dom_component_safe_step local.get_component_impl local.get_dom_component_ptr
        select_result_I2)
qed
end

interpretation i_get_component_get_element_by_id?: l_get_component_get_element_by_id_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_component_get_element_by_id_Core_DOM_def instances)
declare l_get_component_get_element_by_id_Core_DOM_axioms [instances]
```

2.1.5 get_elements_by_class_name

```
locale l_get_component_get_elements_by_class_name_Core_DOM =
  l_get_component_Core_DOM +
  l_first_in_tree_order_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_elements_by_class_name_result_in_tree_order:
  assumes "h ⊢ get_elements_by_class_name ptr name →r results"
  assumes "h ⊢ to_tree_order ptr →r to"
```

```
  assumes "result ∈ set results"
  shows "cast result ∈ set to"
  using assms
  by(auto simp add: get_elements_by_class_name_def first_in_tree_order_def
      elim!: map_filter_M_pure_E[where y=result]  bind_returns_result_E2
      dest!: bind_returns_result_E3[rotated, OF assms(2), rotated]
      intro!: map_filter_M_pure map_M_pure_I bind_pure_I
      split: option.splits list.splits if_splits)

lemma get_elements_by_class_name_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ get_elements_by_class_name ptr' name →r results"
  assumes "result ∈ set results"
  shows "cast result ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "cast result ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    by (metis assms(4) bind_is_OK_E get_component_def is_OK_returns_result_I)
  then have "h ⊢ to_tree_order root_ptr →r c"
    using ‹h ⊢ get_component ptr →r c›
    by (simp add: get_component_def bind_returns_result_E3)

  obtain to' where to': "h ⊢ to_tree_order ptr' →r to'"
    using ‹h ⊢ get_elements_by_class_name ptr' name →r results›
    apply(simp add: get_elements_by_class_name_def first_in_tree_order_def)
    by (meson bind_is_OK_E is_OK_returns_result_I)
  then have "cast result ∈ set to'"
    using get_elements_by_class_name_result_in_tree_order assms by blast

  have "h ⊢ get_root_node (cast result) →r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →r root_ptr"
    using ‹cast result ∈ set to'› ‹h ⊢ to_tree_order ptr' →r to'›
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_ptr get_component_root_node_same
      get_component_subset get_component_to_tree_order
    by blast
  then have "h ⊢ get_component ptr' →r c"
    using ‹h ⊢ to_tree_order root_ptr →r c›
    using get_component_def by auto
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume "ptr' ∈ set c"
  moreover obtain to' where to': "h ⊢ to_tree_order ptr' →r to'"
    by (meson assms(1) assms(2) assms(3) assms(4) calculation get_dom_component_ptr_in_heap
        get_component_subset is_OK_returns_result_E is_OK_returns_result_I to_tree_order_ok)
  ultimately have "set to' ⊆ set c"
    using assms(1) assms(2) assms(3) assms(4) get_component_subset get_component_to_tree_order_subset
    by blast
  moreover have "cast result ∈ set to'"
    using assms get_elements_by_class_name_result_in_tree_order to' by blast
  ultimately show "cast result ∈ set c"
    by blast
qed

lemma get_elements_by_class_name_pure [simp]:
  "pure (get_elements_by_class_name ptr name) h"
  by(auto simp add: get_elements_by_class_name_def intro!: bind_pure_I map_filter_M_pure
      split: option.splits)
```

```
lemma get_elements_by_class_name_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_elements_by_class_name ptr name →_r results"
  assumes "h ⊢ get_elements_by_class_name ptr name →_h h'"
  shows "is_strongly_dom_component_safe {ptr} (cast ` set results) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson get_elements_by_class_name_pure pure_returns_heap_eq)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_elements_by_class_name_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
        local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →_r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
        local.to_tree_order_ok)
  then have "cast ` set results ⊆ set to"
    using assms(4) local.get_elements_by_class_name_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_component ptr →_r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_component_impl local.get_component_ok
    by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def
        get_elements_by_class_name_def first_in_tree_order_def elim!: bind_returns_result_E2
        intro!: map_filter_M_pure bind_pure_I split: option.splits list.splits)[1]
    by (metis (no_types, opaque_lifting) Int_iff ‹ptr |∈| object_ptr_kinds h› assms(4)
        get_elements_by_class_name_is_strongly_dom_component_safe_step local.get_component_impl
        local.get_dom_component_ptr select_result_I2)
qed
end

interpretation i_get_component_get_elements_by_class_name?: l_get_component_get_elements_by_class_name_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_component_get_elements_by_class_name_Core_DOM_def instances)
declare l_get_component_get_elements_by_class_name_Core_DOM_axioms [instances]
```

2.1.6 get_elements_by_tag_name

```
locale l_get_component_get_elements_by_tag_name_Core_DOM =
  l_get_component_Core_DOM +
  l_first_in_tree_order_Core_DOM +
  l_to_tree_order_Core_DOM +
  l_get_element_by_Core_DOM
begin

lemma get_elements_by_tag_name_result_in_tree_order:
  assumes "h ⊢ get_elements_by_tag_name ptr name →_r results"
  assumes "h ⊢ to_tree_order ptr →_r to"
  assumes "result ∈ set results"
  shows "cast result ∈ set to"
  using assms
  by(auto simp add: get_elements_by_tag_name_def first_in_tree_order_def
      elim!: map_filter_M_pure_E[where y=result]  bind_returns_result_E2
      dest!: bind_returns_result_E3[rotated, OF assms(2), rotated]
      intro!: map_filter_M_pure map_M_pure_I bind_pure_I
      split: option.splits list.splits if_splits)
```

```
lemma get_elements_by_tag_name_is_strongly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_component ptr →r c"
  assumes "h ⊢ get_elements_by_tag_name ptr' name →r results"
  assumes "result ∈ set results"
  shows "cast result ∈ set c ⟷ ptr' ∈ set c"
proof
  assume 1: "cast result ∈ set c"
  obtain root_ptr where
    root_ptr: "h ⊢ get_root_node ptr →r root_ptr"
    by (metis assms(4) bind_is_OK_E get_component_def is_OK_returns_result_I)
  then have "h ⊢ to_tree_order root_ptr →r c"
    using ‹h ⊢ get_component ptr →r c›
    by (simp add: get_component_def bind_returns_result_E3)

  obtain to' where to': "h ⊢ to_tree_order ptr' →r to'"
    using ‹h ⊢ get_elements_by_tag_name ptr' name →r results›
    apply(simp add: get_elements_by_tag_name_def first_in_tree_order_def)
    by (meson bind_is_OK_E is_OK_returns_result_I)
  then have "cast result ∈ set to'"
    using get_elements_by_tag_name_result_in_tree_order assms by blast

  have "h ⊢ get_root_node (cast result) →r root_ptr"
    using "1" assms(1) assms(2) assms(3) assms(4) get_component_root_node_same root_ptr
    by blast
  then have "h ⊢ get_root_node ptr' →r root_ptr"
    using ‹cast result ∈ set to'› ‹h ⊢ to_tree_order ptr' →r to'›
    using "1" assms(1) assms(2) assms(3) assms(4) get_dom_component_ptr get_component_root_node_same
      get_component_subset get_component_to_tree_order
    by blast
  then have "h ⊢ get_component ptr' →r c"
    using ‹h ⊢ to_tree_order root_ptr →r c›
    using get_component_def by auto
  then show "ptr' ∈ set c"
    using assms(1) assms(2) assms(3) get_dom_component_ptr by blast
next
  assume "ptr' ∈ set c"
  moreover obtain to' where to': "h ⊢ to_tree_order ptr' →r to'"
    by (meson assms(1) assms(2) assms(3) assms(4) calculation get_dom_component_ptr_in_heap
        get_component_subset is_OK_returns_result_E is_OK_returns_result_I to_tree_order_ok)
  ultimately have "set to' ⊆ set c"
    using assms(1) assms(2) assms(3) assms(4) get_component_subset get_component_to_tree_order_subset
    by blast
  moreover have "cast result ∈ set to'"
    using assms get_elements_by_tag_name_result_in_tree_order to' by blast
  ultimately show "cast result ∈ set c"
    by blast
qed

lemma get_elements_by_tag_name_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_elements_by_tag_name ptr name →r results"
  assumes "h ⊢ get_elements_by_tag_name ptr name →h h'"
  shows "is_strongly_dom_component_safe {ptr} (cast ` set results) h h'"
proof -
  have "h = h'"
    using assms(5)
    by (meson get_elements_by_tag_name_pure pure_returns_heap_eq)
  have "ptr |∈| object_ptr_kinds h"
    using assms(4)
    apply(auto simp add: get_elements_by_tag_name_def)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) bind_is_OK_E is_OK_returns_result_I
```

```
      local.first_in_tree_order_def local.to_tree_order_ptr_in_result local.to_tree_order_ptrs_in_heap)
  obtain to where to: "h ⊢ to_tree_order ptr →r to"
    by (meson ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) is_OK_returns_result_E
      local.to_tree_order_ok)
  then have "cast ` set results ⊆ set to"
    using assms(4) local.get_elements_by_tag_name_result_in_tree_order by auto
  obtain c where c: "h ⊢ a_get_component ptr →r c"
    using ‹ptr |∈| object_ptr_kinds h› assms(1) assms(2) assms(3) local.get_component_impl local.get_component_ok
    by blast

  then show ?thesis
    using assms ‹h = h'›
    apply(auto simp add: is_strongly_dom_component_safe_def Let_def preserved_def
      get_elements_by_class_name_def first_in_tree_order_def elim!: bind_returns_result_E2
      intro!: map_filter_M_pure bind_pure_I split: option.splits list.splits)[1]
    by (metis (no_types, opaque_lifting) IntI ‹ptr |∈| object_ptr_kinds h›
      get_elements_by_tag_name_is_strongly_dom_component_safe_step local.get_component_impl
      local.get_dom_component_ptr select_result_I2)
qed
end

interpretation i_get_component_get_elements_by_tag_name?: l_get_component_get_elements_by_tag_name_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name first_in_tree_order get_attribute get_attribute_locs
  by(auto simp add: l_get_component_get_elements_by_tag_name_Core_DOM_def instances)
declare l_get_component_get_elements_by_tag_name_Core_DOM_axioms [instances]
```

2.1.7 remove_child

```
lemma remove_child_not_strongly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and ptr and child where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ remove_child ptr child →h h'" and
    "¬ is_strongly_dom_component_safe { ptr, cast child} {} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      e2 ← create_element document_ptr ''div'';
      append_child (cast e1) (cast e2);
      return (e1, e2)
  }"
  let ?h1 = "|?h0 ⊢ ?P|h"
  let ?e1 = "fst |?h0 ⊢ ?P|r"
  let ?e2 = "snd |?h0 ⊢ ?P|r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e1" and child="cast_element_ptr2node_ptr
?e2"])
    by code_simp+
qed
```

2.1.8 adopt_node

lemma *adopt_node_not_strongly_dom_component_safe:*
obtains
h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" **and**
h' **and** *document_ptr* **and** *child* **where**
"heap_is_wellformed h" **and** *"type_wf h"* **and** *"known_ptrs h"* **and**
"h ⊢ adopt_node document_ptr child →$_h$ h'" **and**
"¬ is_strongly_dom_component_safe {cast document_ptr, cast child} {} h h'"
proof -
let *?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"*
let *?P = "do {*
document_ptr ← create_document;
document_ptr2 ← create_document;
e1 ← create_element document_ptr ''div'';
adopt_node document_ptr2 (cast e1);
return (document_ptr, e1)
}"
let *?h1 = "|?h0 ⊢ ?P|$_h$"*
let *?document_ptr = "fst |?h0 ⊢ ?P|$_r$"*
let *?e1 = "snd |?h0 ⊢ ?P|$_r$"*

show *thesis*
apply(*rule that*[**where** *h="?h1"* **and** *document_ptr="?document_ptr"* **and** *child="cast*$_{element\_ptr2node\_ptr}$ *?e1"*])
by *code_simp+*
qed

2.1.9 create_element

lemma *create_element_not_strongly_component_safe:*
obtains
h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" **and**
h' **and** *document_ptr* **and** *new_element_ptr* **and** *tag* **where**
"heap_is_wellformed h" **and** *"type_wf h"* **and** *"known_ptrs h"* **and**
"h ⊢ create_element document_ptr tag →$_r$ new_element_ptr →$_h$ h'" **and**
"¬ is_strongly_dom_component_safe {cast document_ptr} {cast new_element_ptr} h h'"
proof -
let *?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"*
let *?P = "create_document"*
let *?h1 = "|?h0 ⊢ ?P|$_h$"*
let *?document_ptr = "|?h0 ⊢ ?P|$_r$"*

show *thesis*
apply(*rule that*[**where** *h="?h1"* **and** *document_ptr="?document_ptr"*])
by *code_simp+*
qed

locale *l_get_component_create_element*$_{Core\_DOM}$ =
l_get_component$_{Core\_DOM}$ *heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order*

```
  get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name +
  l_create_element_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs set_tag_name set_tag_name_locs type_wf create_element known_ptr +
  l_get_disconnected_nodes_Core_DOM type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_disconnected_nodes_Core_DOM type_wf set_disconnected_nodes set_disconnected_nodes_locs +
  l_set_tag_name_Core_DOM type_wf set_tag_name set_tag_name_locs +
  l_new_element_get_child_nodes_Core_DOM type_wf known_ptr get_child_nodes get_child_nodes_locs +
  l_new_element_get_disconnected_nodes get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_tag_name_get_child_nodes type_wf set_tag_name set_tag_name_locs known_ptr
  get_child_nodes get_child_nodes_locs +
  l_set_tag_name_get_disconnected_nodes type_wf set_tag_name set_tag_name_locs
  get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_disconnected_nodes type_wf set_disconnected_nodes set_disconnected_nodes_locs +
  l_set_disconnected_nodes_get_child_nodes set_disconnected_nodes set_disconnected_nodes_locs
  get_child_nodes get_child_nodes_locs +
  l_set_disconnected_nodes_get_disconnected_nodes  type_wf get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs +
  l_new_element type_wf +
  l_create_element_wf_Core_DOM  known_ptr known_ptrs type_wf get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_nodes_locs heap_is_wellformed parent_child_rel set_tag_name
  set_tag_name_locs set_disconnected_nodes set_disconnected_nodes_locs
  create_element
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and type_wf :: "(_) heap ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and is_strongly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and is_weakly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option)
prog"
    and get_elements_by_class_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    and get_elements_by_tag_name :: "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr
list) prog"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_tag_name :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
    and set_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and create_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
begin

lemma create_element_is_weakly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_element document_ptr tag →_h h'"
```

```
  assumes "ptr ∉ set |h ⊢ get_component (cast document_ptr)|_r"
  assumes "ptr ≠ cast |h ⊢ create_element document_ptr tag|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain new_element_ptr h2 h3 disc_nodes where
    new_element_ptr: "h ⊢ new_element →_r new_element_ptr" and
    h2: "h ⊢ new_element →_h h2" and
    h3: "h2 ⊢set_tag_name new_element_ptr tag →_h h3" and
    disc_nodes: "h3 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_element_ptr # disc_nodes) →_h h'"
    using assms(4)
    by(auto simp add: create_element_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated])
  have "h ⊢ create_element document_ptr tag →_r new_element_ptr"
    using new_element_ptr h2 h3 disc_nodes h'
    apply(auto simp add: create_element_def intro!: bind_returns_result_I
        bind_pure_returns_result_I[OF get_disconnected_nodes_pure])[1]
     apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
  have "preserved (get_M ptr getter) h h2"
    using h2 new_element_ptr
    apply(rule new_element_get_M_Object)
    using new_element_ptr assms(6) ‹h ⊢ create_element document_ptr tag →_r new_element_ptr›
    by simp

  have "preserved (get_M ptr getter) h2 h3"
    using set_tag_name_writes h3
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_tag_name_locs_impl a_set_tag_name_locs_def all_args_def)[1]
    by (metis (no_types, lifting) ‹h ⊢ create_element document_ptr tag →_r new_element_ptr› assms(6)
        get_M_Element_preserved8 select_result_I2)

  have "document_ptr |∈| document_ptr_kinds h"
    using create_element_document_in_heap
    using assms(4)
    by blast
  then
  have "ptr ≠ (cast document_ptr)"
    using assms(5) assms(1) assms(2) assms(3) local.get_component_ok local.get_dom_component_ptr
    by auto
  have "preserved (get_M ptr getter) h3 h'"
    using set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_disconnected_nodes_locs_def all_args_def)[1]
    by (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)

  show ?thesis
    using ‹preserved (get_M ptr getter) h h2› ‹preserved (get_M ptr getter) h2 h3›
      ‹preserved (get_M ptr getter) h3 h'›
    by(auto simp add: preserved_def)
qed


lemma create_element_is_weakly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_element document_ptr tag →_r result →_h h'"
  shows "is_weakly_dom_component_safe {cast document_ptr} {cast result} h h'"
proof -
  obtain new_element_ptr h2 h3 disc_nodes_h3 where
    new_element_ptr: "h ⊢ new_element →_r new_element_ptr" and
    h2: "h ⊢ new_element →_h h2" and
    h3: "h2 ⊢ set_tag_name new_element_ptr tag →_h h3" and
    disc_nodes_h3: "h3 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes_h3" and
```

```
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_element_ptr # disc_nodes_h3) →h h'"
    using assms(4)
    by(auto simp add: create_element_def returns_result_heap_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated] )
  then have "h ⊢ create_element document_ptr tag →r new_element_ptr"
    apply(auto simp add: create_element_def intro!: bind_returns_result_I)[1]
     apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
    apply (metis is_OK_returns_heap_E is_OK_returns_result_I local.get_disconnected_nodes_pure
        pure_returns_heap_eq)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
  have "h ⊢ create_element document_ptr tag →h h'"
    by (meson assms(4) returns_result_heap_def)


  have "new_element_ptr ∉ set |h ⊢ element_ptr_kinds_M|r"
    using new_element_ptr ElementMonad.ptr_kinds_ptr_kinds_M h2
    using new_element_ptr_not_in_heap by blast
  then have "cast new_element_ptr ∉ set |h ⊢ node_ptr_kinds_M|r"
    by simp
  then have "cast new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|r"
    by simp

  have object_ptr_kinds_eq_h: "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_element_ptr|}"
    using new_element_new_ptr h2 new_element_ptr by blast
  then have node_ptr_kinds_eq_h: "node_ptr_kinds h2 = node_ptr_kinds h |∪| {|cast new_element_ptr|}"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h |∪| {|new_element_ptr|}"
    apply(simp add: element_ptr_kinds_def)
    by force
  have character_data_ptr_kinds_eq_h: "character_data_ptr_kinds h2 = character_data_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: node_ptr_kinds_def character_data_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h2 = document_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: document_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_tag_name_writes h3])
    using set_tag_name_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_disconnected_nodes_writes h'])
    using set_disconnected_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
    using object_ptr_kinds_eq_h3
    by(auto simp add: node_ptr_kinds_def)

  have "known_ptr (cast new_element_ptr)"
    using ‹h ⊢ create_element document_ptr tag →r new_element_ptr› local.create_element_known_ptr
    by blast
```

```
then
have "known_ptrs h2"
  using known_ptrs_new_ptr object_ptr_kinds_eq_h ‹known_ptrs h› h2
  by blast
then
have "known_ptrs h3"
  using known_ptrs_preserved object_ptr_kinds_eq_h2 by blast
then
have "known_ptrs h'"
  using known_ptrs_preserved object_ptr_kinds_eq_h3 by blast


have "document_ptr |∈| document_ptr_kinds h"
  using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
    get_disconnected_nodes_ptr_in_heap ‹type_wf h› document_ptr_kinds_def
  by (metis is_OK_returns_result_I)

have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_element_ptr
           ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
  using get_child_nodes_reads h2 get_child_nodes_new_element[rotated, OF new_element_ptr h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by blast+
then have children_eq2_h: "⋀ptr'. ptr' ≠ cast new_element_ptr
                                ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
  using select_result_eq by force

have "h2 ⊢ get_child_nodes (cast new_element_ptr) →r []"
  using new_element_ptr h2 new_element_ptr_in_heap[OF h2 new_element_ptr]
    new_element_is_element_ptr[OF new_element_ptr] new_element_no_child_nodes
  by blast
have disconnected_nodes_eq_h:
  "⋀doc_ptr disc_nodes. h ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                        = h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
  using get_disconnected_nodes_reads h2 get_disconnected_nodes_new_element[OF new_element_ptr h2]
  apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
  by blast+
then have disconnected_nodes_eq2_h:
  "⋀doc_ptr. |h ⊢ get_disconnected_nodes doc_ptr|r = |h2 ⊢ get_disconnected_nodes doc_ptr|r"
  using select_result_eq by force

have children_eq_h2:
  "⋀ptr' children. h2 ⊢ get_child_nodes ptr' →r children = h3 ⊢ get_child_nodes ptr' →r children"
  using get_child_nodes_reads set_tag_name_writes h3
  apply(rule reads_writes_preserved)
  by(auto simp add: set_tag_name_get_child_nodes)
then have children_eq2_h2: "⋀ptr'. |h2 ⊢ get_child_nodes ptr'|r = |h3 ⊢ get_child_nodes ptr'|r"
  using select_result_eq by force
have disconnected_nodes_eq_h2:
  "⋀doc_ptr disc_nodes. h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                          = h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
  using get_disconnected_nodes_reads set_tag_name_writes h3
  apply(rule reads_writes_preserved)
  by(auto simp add: set_tag_name_get_disconnected_nodes)
then have disconnected_nodes_eq2_h2:
  "⋀doc_ptr. |h2 ⊢ get_disconnected_nodes doc_ptr|r = |h3 ⊢ get_disconnected_nodes doc_ptr|r"
  using select_result_eq by force

have "type_wf h2"
  using ‹type_wf h› new_element_types_preserved h2 by blast
then have "type_wf h3"
  using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_tag_name_writes h3]
  using  set_tag_name_types_preserved
  by(auto simp add: reflp_def transp_def)
```

```
  then have "type_wf h'"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h']
    using  set_disconnected_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)

  have children_eq_h3:
    "⋀ptr' children. h3 ⊢ get_child_nodes ptr' →r children = h' ⊢ get_child_nodes ptr' →r children"
    using get_child_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_child_nodes)
  then have children_eq2_h3: "⋀ptr'. |h3 ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have disconnected_nodes_eq_h3:
    "⋀doc_ptr disc_nodes. document_ptr ≠ doc_ptr
       ⟹ h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                          = h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using get_disconnected_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_disconnected_nodes_different_pointers)
  then have disconnected_nodes_eq2_h3:
    "⋀doc_ptr. document_ptr ≠ doc_ptr
               ⟹ |h3 ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force

  have disc_nodes_document_ptr_h2: "h2 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h2 disc_nodes_h3 by auto
  then have disc_nodes_document_ptr_h: "h ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h by auto
  then have "cast new_element_ptr ∉ set disc_nodes_h3"
    using ‹heap_is_wellformed h›
    using ‹cast_element_ptr2node_ptr new_element_ptr ∉ set |h ⊢ node_ptr_kinds_M|r›
      a_all_ptrs_in_heap_def heap_is_wellformed_def
    using NodeMonad.ptr_kinds_ptr_kinds_M local.heap_is_wellformed_disc_nodes_in_heap by blast


  have "parent_child_rel h = parent_child_rel h'"
  proof -
    have "parent_child_rel h = parent_child_rel h2"
    proof(auto simp add: parent_child_rel_def)[1]
      fix a x
      assume 0: "a |∈| object_ptr_kinds h"
        and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
      then show "a |∈| object_ptr_kinds h2"
        by (simp add: object_ptr_kinds_eq_h)
    next
      fix a x
      assume 0: "a |∈| object_ptr_kinds h"
        and 1: "x ∈ set |h ⊢ get_child_nodes a|r"
      then show "x ∈ set |h2 ⊢ get_child_nodes a|r"
        by (metis ObjectMonad.ptr_kinds_ptr_kinds_M
            ‹cast_element_ptr2object_ptr new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|r› children_eq2_h)
    next
      fix a x
      assume 0: "a |∈| object_ptr_kinds h2"
        and 1: "x ∈ set |h2 ⊢ get_child_nodes a|r"
      then show "a |∈| object_ptr_kinds h"
        using object_ptr_kinds_eq_h ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →r
[]›
        by(auto)
    next
      fix a x
      assume 0: "a |∈| object_ptr_kinds h2"
```

```
      and 1: "x ∈ set |h2 ⊢ get_child_nodes a|_r"
    then show "x ∈ set |h ⊢ get_child_nodes a|_r"
      by (metis (no_types, lifting)
          ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →_r []›
          children_eq2_h empty_iff empty_set image_eqI select_result_I2)
  qed
  also have "... = parent_child_rel h3"
    by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
  also have "... = parent_child_rel h'"
    by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
  finally show ?thesis
    by simp
qed
have root: "h ⊢ get_root_node (cast document_ptr) →_r cast document_ptr"
  by (simp add: ‹document_ptr |∈| document_ptr_kinds h› local.get_root_node_not_node_same)
then
have root': "h' ⊢ get_root_node (cast document_ptr) →_r cast document_ptr"
  by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h
      local.get_root_node_not_node_same object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3)


have "heap_is_wellformed h'"
  using create_element_preserves_wellformedness assms returns_result_heap_def
  by metis

have "cast result |∉| object_ptr_kinds h"
  using ‹cast new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r› returns_result_heap_def
  by (metis (full_types) ObjectMonad.ptr_kinds_ptr_kinds_M
      ‹h ⊢ create_element document_ptr tag →_r new_element_ptr› assms(4) returns_result_eq)

obtain to where to: "h ⊢ to_tree_order (cast document_ptr) →_r to"
  by (meson ‹document_ptr |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
      document_ptr_kinds_commutes is_OK_returns_result_E local.to_tree_order_ok)
then
have "h ⊢ a_get_component (cast document_ptr) →_r to"
  using root
  by(auto simp add: a_get_component_def)
moreover
obtain to' where to': "h' ⊢ to_tree_order (cast document_ptr) →_r to'"
  by (meson ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹type_wf h'› is_OK_returns_result_E
      local.get_root_node_root_in_heap local.to_tree_order_ok root')
then
have "h' ⊢ a_get_component (cast document_ptr) →_r to'"
  using root'
  by(auto simp add: a_get_component_def)
moreover
have "⋀child. child ∈ set to ⟷ child ∈ set to'"
  by (metis ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹parent_child_rel h = parent_child_rel h'›
      ‹type_wf h'› assms(1) assms(2) assms(3) local.to_tree_order_parent_child_rel to to')
ultimately
have "set |h ⊢ local.a_get_component (cast document_ptr)|_r = set |h' ⊢ local.a_get_component (cast document_ptr)
  by(auto simp add: a_get_component_def)

show ?thesis
  apply(auto simp add: is_weakly_dom_component_safe_def Let_def)[1]
     apply (metis ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr new_element_ptr) →_r []› assms(2) assms(3)
      children_eq_h local.get_child_nodes_ok local.get_child_nodes_ptr_in_heap local.known_ptrs_known_ptr
      object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 returns_result_select_result)
    apply (metis ‹h ⊢ create_element document_ptr tag →_r new_element_ptr› assms(4)
      element_ptr_kinds_commutes h2 new_element_ptr new_element_ptr_in_heap node_ptr_kinds_eq_h2
      node_ptr_kinds_eq_h3 returns_result_eq returns_result_heap_def)
  using ‹cast_element_ptr2object_ptr result |∉| object_ptr_kinds h› element_ptr_kinds_commutes
    node_ptr_kinds_commutes apply blast
```

```
    using assms(1) assms(2) assms(3) ‹h ⊢ create_element document_ptr tag →_h h'›
    apply(rule create_element_is_weakly_dom_component_safe_step)
     apply (simp add: local.get_component_impl)
    using ‹cast_element_ptr2object_ptr new_element_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r›
      ‹h ⊢ create_element document_ptr tag →_r new_element_ptr› by auto
qed
end

interpretation i_get_component_create_element?: l_get_component_create_element_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs set_tag_name
  set_tag_name_locs create_element
  by(auto simp add: l_get_component_create_element_Core_DOM_def instances)
declare l_get_component_create_element_Core_DOM_axioms [instances]
```

2.1.10 create_character_data

```
lemma create_character_data_not_strongly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and document_ptr and new_character_data_ptr and val where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ create_character_data document_ptr val →_r new_character_data_ptr →_h h'" and
    "¬ is_strongly_dom_component_safe {cast document_ptr} {cast new_character_data_ptr} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "create_document"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?document_ptr = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and document_ptr="?document_ptr"])
    by code_simp+
qed

locale l_get_component_create_character_data_Core_DOM =
  l_get_component_Core_DOM heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order
  get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name +
  l_create_character_data_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs set_val set_val_locs  type_wf create_character_data known_ptr +
  l_get_disconnected_nodes_Core_DOM type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_set_disconnected_nodes_Core_DOM type_wf set_disconnected_nodes set_disconnected_nodes_locs +
  l_set_val_Core_DOM type_wf set_val set_val_locs +
  l_create_character_data_wf_Core_DOM known_ptr type_wf get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_nodes_locs heap_is_wellformed parent_child_rel set_val
  set_val_locs set_disconnected_nodes set_disconnected_nodes_locs
  create_character_data known_ptrs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and type_wf :: "(_) heap ⇒ bool"
```

```
    and known_ptrs :: "(_) heap ⇒ bool"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and is_strongly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and is_weakly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and get_elements_by_class_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and get_elements_by_tag_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and set_val :: "(_) character_data_ptr ⇒ char list ⇒ ((_) heap, exception, unit) prog"
    and set_val_locs :: "(_) character_data_ptr ⇒ ((_) heap, exception, unit) prog set"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_disconnected_nodes :: "(_) document_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap, exception, unit) prog set"
    and create_character_data ::
    "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) character_data_ptr) prog"
begin

lemma create_character_data_is_weakly_dom_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_character_data document_ptr text →_h h'"
  assumes "ptr ∉ set |h ⊢ get_component (cast document_ptr)|_r"
  assumes "ptr ≠ cast |h ⊢ create_character_data document_ptr text|_r"
  shows "preserved (get_M ptr getter) h h'"
proof -
  obtain new_character_data_ptr h2 h3 disc_nodes where
    new_character_data_ptr: "h ⊢ new_character_data →_r new_character_data_ptr" and
    h2: "h ⊢ new_character_data →_h h2" and
    h3: "h2 ⊢ set_val new_character_data_ptr text →_h h3" and
    disc_nodes: "h3 ⊢ get_disconnected_nodes document_ptr →_r disc_nodes" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_character_data_ptr # disc_nodes) →_h h'"
    using assms(4)
    by(auto simp add: create_character_data_def
        elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated])

  have "h ⊢ create_character_data document_ptr text →_r new_character_data_ptr"
    using new_character_data_ptr h2 h3 disc_nodes h'
    apply(auto simp add: create_character_data_def
        intro!: bind_returns_result_I bind_pure_returns_result_I[OF get_disconnected_nodes_pure])[1]
     apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
  have "preserved (get_M ptr getter) h h2"
    using h2 new_character_data_ptr
    apply(rule new_character_data_get_M_Object)
    using new_character_data_ptr assms(6)
      ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr›
    by simp

  have "preserved (get_M ptr getter) h2 h3"
```

```
    using set_val_writes h3
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_val_locs_impl a_set_val_locs_def all_args_def)[1]
    by (metis (mono_tags) CharacterData_simp11
        ‹h ⊢ create_character_data document_ptr text →r new_character_data_ptr› assms(4) assms(6)
        is_OK_returns_heap_I is_OK_returns_result_E returns_result_eq select_result_I2)

  have "document_ptr |∈| document_ptr_kinds h"
    using create_character_data_document_in_heap
    using assms(4)
    by blast
  then
  have "ptr ≠ (cast document_ptr)"
    using assms(5) assms(1) assms(2) assms(3) local.get_component_ok local.get_dom_component_ptr
    by auto
  have "preserved (get_M ptr getter) h3 h'"
    using set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_disconnected_nodes_locs_def all_args_def)[1]
    by (metis ‹ptr ≠ cast_document_ptr2object_ptr document_ptr› get_M_Mdocument_preserved3)

  show ?thesis
    using ‹preserved (get_M ptr getter) h h2› ‹preserved (get_M ptr getter) h2 h3›
      ‹preserved (get_M ptr getter) h3 h'›
    by(auto simp add: preserved_def)
qed

lemma create_character_data_is_weakly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_character_data document_ptr text →r result"
  assumes "h ⊢ create_character_data document_ptr text →h h'"
  shows "is_weakly_dom_component_safe {cast document_ptr} {cast result} h h'"
proof -

  obtain new_character_data_ptr h2 h3 disc_nodes_h3 where
    new_character_data_ptr: "h ⊢ new_character_data →r new_character_data_ptr" and
    h2: "h ⊢ new_character_data →h h2" and
    h3: "h2 ⊢ set_val new_character_data_ptr text →h h3" and
    disc_nodes_h3: "h3 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3" and
    h': "h3 ⊢ set_disconnected_nodes document_ptr (cast new_character_data_ptr # disc_nodes_h3) →h h'"
    using assms(5)
    by(auto simp add: create_character_data_def
        elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_disconnected_nodes_pure, rotated] )
  then
  have "h ⊢ create_character_data document_ptr text →r new_character_data_ptr"
    apply(auto simp add: create_character_data_def intro!: bind_returns_result_I)[1]
      apply (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)
     apply (metis is_OK_returns_heap_E is_OK_returns_result_I local.get_disconnected_nodes_pure
        pure_returns_heap_eq)
    by (metis is_OK_returns_heap_I is_OK_returns_result_E old.unit.exhaust)

  have "new_character_data_ptr ∉ set |h ⊢ character_data_ptr_kinds_M|r"
    using new_character_data_ptr CharacterDataMonad.ptr_kinds_ptr_kinds_M h2
    using new_character_data_ptr_not_in_heap by blast
  then have "cast new_character_data_ptr ∉ set |h ⊢ node_ptr_kinds_M|r"
    by simp
  then have "cast new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|r"
    by simp



  have object_ptr_kinds_eq_h:
```

```
    "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    using new_character_data_new_ptr h2 new_character_data_ptr by blast
  then have node_ptr_kinds_eq_h:
    "node_ptr_kinds h2 = node_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have character_data_ptr_kinds_eq_h:
    "character_data_ptr_kinds h2 = character_data_ptr_kinds h |∪| {|new_character_data_ptr|}"
    apply(simp add: character_data_ptr_kinds_def)
    by force
  have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: node_ptr_kinds_def element_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h2 = document_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: document_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_val_writes h3])
    using set_val_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_disconnected_nodes_writes h'])
    using set_disconnected_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
    using object_ptr_kinds_eq_h3
    by(auto simp add: node_ptr_kinds_def)


  have "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      get_disconnected_nodes_ptr_in_heap <type_wf h> document_ptr_kinds_def
    by (metis is_OK_returns_result_I)

  have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_character_data_ptr
                ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using get_child_nodes_reads h2 get_child_nodes_new_character_data[rotated, OF new_character_data_ptr
h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have children_eq2_h:
    "⋀ptr'. ptr' ≠ cast new_character_data_ptr
      ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have object_ptr_kinds_eq_h:
    "object_ptr_kinds h2 = object_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    using new_character_data_new_ptr h2 new_character_data_ptr by blast
  then have node_ptr_kinds_eq_h:
    "node_ptr_kinds h2 = node_ptr_kinds h |∪| {|cast new_character_data_ptr|}"
    apply(simp add: node_ptr_kinds_def)
    by force
  then have character_data_ptr_kinds_eq_h:
```

```
    "character_data_ptr_kinds h2 = character_data_ptr_kinds h |∪| {|new_character_data_ptr|}"
    apply(simp add: character_data_ptr_kinds_def)
    by force
  have element_ptr_kinds_eq_h: "element_ptr_kinds h2 = element_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: node_ptr_kinds_def element_ptr_kinds_def)
  have document_ptr_kinds_eq_h: "document_ptr_kinds h2 = document_ptr_kinds h"
    using object_ptr_kinds_eq_h
    by(auto simp add: document_ptr_kinds_def)

  have object_ptr_kinds_eq_h2: "object_ptr_kinds h3 = object_ptr_kinds h2"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_val_writes h3])
    using set_val_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h2: "document_ptr_kinds h3 = document_ptr_kinds h2"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h2: "node_ptr_kinds h3 = node_ptr_kinds h2"
    using object_ptr_kinds_eq_h2
    by(auto simp add: node_ptr_kinds_def)

  have object_ptr_kinds_eq_h3: "object_ptr_kinds h' = object_ptr_kinds h3"
    apply(rule writes_small_big[where P="λh h'. object_ptr_kinds h' = object_ptr_kinds h",
          OF set_disconnected_nodes_writes h'])
    using set_disconnected_nodes_pointers_preserved
    by (auto simp add: reflp_def transp_def)
  then have document_ptr_kinds_eq_h3: "document_ptr_kinds h' = document_ptr_kinds h3"
    by (auto simp add: document_ptr_kinds_def)
  have node_ptr_kinds_eq_h3: "node_ptr_kinds h' = node_ptr_kinds h3"
    using object_ptr_kinds_eq_h3
    by(auto simp add: node_ptr_kinds_def)


  have "document_ptr |∈| document_ptr_kinds h"
    using disc_nodes_h3 document_ptr_kinds_eq_h object_ptr_kinds_eq_h2
      get_disconnected_nodes_ptr_in_heap <type_wf h> document_ptr_kinds_def
    by (metis is_OK_returns_result_I)

  have children_eq_h: "⋀(ptr'::(_) object_ptr) children. ptr' ≠ cast new_character_data_ptr
              ⟹ h ⊢ get_child_nodes ptr' →r children = h2 ⊢ get_child_nodes ptr' →r children"
    using get_child_nodes_reads h2
      get_child_nodes_new_character_data[rotated, OF new_character_data_ptr h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have children_eq2_h: "⋀ptr'. ptr' ≠ cast new_character_data_ptr
                              ⟹ |h ⊢ get_child_nodes ptr'|r = |h2 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force

  have "h2 ⊢ get_child_nodes (cast new_character_data_ptr) →r []"
    using new_character_data_ptr h2 new_character_data_ptr_in_heap[OF h2 new_character_data_ptr]
      new_character_data_is_character_data_ptr[OF new_character_data_ptr]
      new_character_data_no_child_nodes
    by blast
  have disconnected_nodes_eq_h:
    "⋀doc_ptr disc_nodes. h ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                        = h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using get_disconnected_nodes_reads h2
      get_disconnected_nodes_new_character_data[OF new_character_data_ptr h2]
    apply(auto simp add: reads_def reflp_def transp_def preserved_def)[1]
    by blast+
  then have disconnected_nodes_eq2_h:
    "⋀doc_ptr. |h ⊢ get_disconnected_nodes doc_ptr|r = |h2 ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force
```

```
  have children_eq_h2:
    "⋀ptr' children. h2 ⊢ get_child_nodes ptr' →r children = h3 ⊢ get_child_nodes ptr' →r children"
    using get_child_nodes_reads set_val_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_val_get_child_nodes)
  then have children_eq2_h2:
    "⋀ptr'. |h2 ⊢ get_child_nodes ptr'|r = |h3 ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have disconnected_nodes_eq_h2:
    "⋀doc_ptr disc_nodes. h2 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                                          = h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using get_disconnected_nodes_reads set_val_writes h3
    apply(rule reads_writes_preserved)
    by(auto simp add: set_val_get_disconnected_nodes)
  then have disconnected_nodes_eq2_h2:
    "⋀doc_ptr. |h2 ⊢ get_disconnected_nodes doc_ptr|r = |h3 ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force

  have "type_wf h2"
    using ‹type_wf h› new_character_data_types_preserved h2 by blast
  then have "type_wf h3"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_val_writes h3]
    using  set_val_types_preserved
    by(auto simp add: reflp_def transp_def)
  then have "type_wf h'"
    using writes_small_big[where P="λh h'. type_wf h ⟶ type_wf h'", OF set_disconnected_nodes_writes
h']
    using  set_disconnected_nodes_types_preserved
    by(auto simp add: reflp_def transp_def)

  have children_eq_h3:
    "⋀ptr' children. h3 ⊢ get_child_nodes ptr' →r children = h' ⊢ get_child_nodes ptr' →r children"
    using get_child_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_child_nodes)
  then have children_eq2_h3:
    " ⋀ptr'. |h3 ⊢ get_child_nodes ptr'|r = |h' ⊢ get_child_nodes ptr'|r"
    using select_result_eq by force
  have disconnected_nodes_eq_h3: "⋀doc_ptr disc_nodes. document_ptr ≠ doc_ptr
    ⟹ h3 ⊢ get_disconnected_nodes doc_ptr →r disc_nodes
                          = h' ⊢ get_disconnected_nodes doc_ptr →r disc_nodes"
    using get_disconnected_nodes_reads set_disconnected_nodes_writes h'
    apply(rule reads_writes_preserved)
    by(auto simp add: set_disconnected_nodes_get_disconnected_nodes_different_pointers)
  then have disconnected_nodes_eq2_h3: "⋀doc_ptr. document_ptr ≠ doc_ptr
           ⟹ |h3 ⊢ get_disconnected_nodes doc_ptr|r = |h' ⊢ get_disconnected_nodes doc_ptr|r"
    using select_result_eq by force

  have disc_nodes_document_ptr_h2: "h2 ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h2 disc_nodes_h3 by auto
  then have disc_nodes_document_ptr_h: "h ⊢ get_disconnected_nodes document_ptr →r disc_nodes_h3"
    using disconnected_nodes_eq_h by auto
  then have "cast new_character_data_ptr ∉ set disc_nodes_h3"
    using ‹heap_is_wellformed h› using ‹cast new_character_data_ptr ∉ set |h ⊢ node_ptr_kinds_M|r›
      a_all_ptrs_in_heap_def heap_is_wellformed_def
    using NodeMonad.ptr_kinds_ptr_kinds_M local.heap_is_wellformed_disc_nodes_in_heap by blast

  have "parent_child_rel h = parent_child_rel h'"
  proof -
    have "parent_child_rel h = parent_child_rel h2"
    proof(auto simp add: parent_child_rel_def)[1]
      fix a x
```

```
    assume 0: "a |∈| object_ptr_kinds h"
      and 1: "x ∈ set |h ⊢ get_child_nodes a|_r"
    then show "a |∈| object_ptr_kinds h2"
      by (simp add: object_ptr_kinds_eq_h)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h"
      and 1: "x ∈ set |h ⊢ get_child_nodes a|_r"
    then show "x ∈ set |h2 ⊢ get_child_nodes a|_r"
      by (metis ObjectMonad.ptr_kinds_ptr_kinds_M
          ‹cast new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r› children_eq2_h)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h2"
      and 1: "x ∈ set |h2 ⊢ get_child_nodes a|_r"
    then show "a |∈| object_ptr_kinds h"
      using object_ptr_kinds_eq_h ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →_r []›
      by(auto)
  next
    fix a x
    assume 0: "a |∈| object_ptr_kinds h2"
      and 1: "x ∈ set |h2 ⊢ get_child_nodes a|_r"
    then show "x ∈ set |h ⊢ get_child_nodes a|_r"
      by (metis (no_types, lifting) ‹h2 ⊢ get_child_nodes (cast new_character_data_ptr) →_r []›
          children_eq2_h empty_iff empty_set image_eqI select_result_I2)
  qed
  also have "... = parent_child_rel h3"
    by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h2 children_eq2_h2)
  also have "... = parent_child_rel h'"
    by(auto simp add: parent_child_rel_def object_ptr_kinds_eq_h3 children_eq2_h3)
  finally show ?thesis
    by simp
qed
have root: "h ⊢ get_root_node (cast document_ptr) →_r cast document_ptr"
  by (simp add: ‹document_ptr |∈| document_ptr_kinds h› local.get_root_node_not_node_same)
then
have root': "h' ⊢ get_root_node (cast document_ptr) →_r cast document_ptr"
  by (simp add: ‹document_ptr |∈| document_ptr_kinds h› document_ptr_kinds_eq_h
      local.get_root_node_not_node_same object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3)


have "heap_is_wellformed h'" and "known_ptrs h'"
  using create_character_data_preserves_wellformedness assms
  by blast+

have "cast result |∉| object_ptr_kinds h"
  using ‹cast new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r›
  by (metis (full_types) ObjectMonad.ptr_kinds_ptr_kinds_M
      ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr› assms(4) returns_result_eq)

obtain to where to: "h ⊢ to_tree_order (cast document_ptr) →_r to"
  by (meson ‹document_ptr |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
      document_ptr_kinds_commutes is_OK_returns_result_E local.to_tree_order_ok)
then
have "h ⊢ a_get_component (cast document_ptr) →_r to"
  using root
  by(auto simp add: a_get_component_def)
moreover
obtain to' where to': "h' ⊢ to_tree_order (cast document_ptr) →_r to'"
  by (meson ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹type_wf h'› is_OK_returns_result_E
      local.get_root_node_root_in_heap local.to_tree_order_ok root')
then
have "h' ⊢ a_get_component (cast document_ptr) →_r to'"
```

```
    using root'
    by(auto simp add: a_get_component_def)
  moreover
  have "⋀child. child ∈ set to ⟷ child ∈ set to'"
    by (metis ‹heap_is_wellformed h'› ‹known_ptrs h'› ‹parent_child_rel h = parent_child_rel h'›
        ‹type_wf h'› assms(1) assms(2) assms(3) local.to_tree_order_parent_child_rel to to')
  ultimately
  have "set |h ⊢ local.a_get_component (cast document_ptr)|_r =
set |h' ⊢ local.a_get_component (cast document_ptr)|_r"
    by(auto simp add: a_get_component_def)

  show ?thesis
    apply(auto simp add: is_weakly_dom_component_safe_def Let_def)[1]
    using  assms(2) assms(3) children_eq_h local.get_child_nodes_ok local.get_child_nodes_ptr_in_heap
      local.known_ptrs_known_ptr object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 returns_result_select_result
     apply (metis ‹h2 ⊢ get_child_nodes (cast_character_data_ptr2object_ptr new_character_data_ptr) →_r []›
      is_OK_returns_result_I)

      apply (metis ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr› assms(4)
        character_data_ptr_kinds_commutes h2 new_character_data_ptr new_character_data_ptr_in_heap
        node_ptr_kinds_eq_h2 node_ptr_kinds_eq_h3 returns_result_eq)
    using ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr›
      ‹new_character_data_ptr ∉ set |h ⊢ character_data_ptr_kinds_M|_r› assms(4) returns_result_eq
     apply fastforce
    using  assms(2) assms(3) children_eq_h local.get_child_nodes_ok local.get_child_nodes_ptr_in_heap
      local.known_ptrs_known_ptr object_ptr_kinds_eq_h2 object_ptr_kinds_eq_h3 returns_result_select_result
    apply (smt ObjectMonad.ptr_kinds_ptr_kinds_M
        ‹cast_character_data_ptr2object_ptr new_character_data_ptr ∉ set |h ⊢ object_ptr_kinds_M|_r›
        ‹h ⊢ create_character_data document_ptr text →_r new_character_data_ptr› assms(1) assms(5)
        create_character_data_is_weakly_dom_component_safe_step local.get_component_impl select_result_I2)
    done
qed
end

interpretation i_get_component_create_character_data?: l_get_component_create_character_data_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name set_val set_val_locs
  get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs
  create_character_data
  by(auto simp add: l_get_component_create_character_data_Core_DOM_def instances)
declare l_get_component_create_character_data_Core_DOM_axioms [instances]
```

2.1.11 create_document

```
lemma create_document_not_strongly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and new_document_ptr where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ create_document →_r new_document_ptr →_h h'" and
    "¬ is_strongly_dom_component_safe {} {cast new_document_ptr} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "create_document"
  let ?h1 = "|?h0 ⊢ ?P|_h"
```

```
  let ?document_ptr = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1"])
    by code_simp+
qed

locale l_get_component_create_document_Core_DOM =
  l_get_component_Core_DOM heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order
  get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name +
  l_create_document_Core_DOM create_document
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and type_wf :: "(_) heap ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and is_strongly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and is_weakly_dom_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and get_elements_by_class_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and get_elements_by_tag_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and create_document :: "((_) heap, exception, (_) document_ptr) prog"
begin

lemma create_document_is_weakly_component_safe_step:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_document →_h h'"
  assumes "ptr ≠ cast |h ⊢ create_document|_r"
  shows "preserved (get_M_Object ptr getter) h h'"
  using assms
  apply(auto simp add: create_document_def)[1]
  by (metis assms(4) assms(5) is_OK_returns_heap_I local.create_document_def new_document_get_M_Object
      select_result_I)

lemma create_document_is_weakly_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ create_document →_r result"
  assumes "h ⊢ create_document →_h h'"
  shows "is_weakly_dom_component_safe {} {cast result} h h'"
proof -
  have "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast result|}"
    using assms(4) assms(5) local.create_document_def new_document_new_ptr by auto
  moreover have "result |∉| document_ptr_kinds h"
    using assms(4) assms(5) local.create_document_def new_document_ptr_not_in_heap by auto
  ultimately show ?thesis
```

```
    using assms
    apply(auto simp add: is_weakly_dom_component_safe_def Let_def local.create_document_def
        new_document_ptr_not_in_heap)[1]
    by (metis ‹result |∉| document_ptr_kinds h› document_ptr_kinds_commutes new_document_get_M_Object)
qed
end

interpretation i_get_component_create_document?: l_get_component_create_document_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name create_document
  by(auto simp add: l_get_component_create_document_Core_DOM_def instances)
declare l_get_component_create_document_Core_DOM_axioms [instances]
```

2.1.12 insert_before

```
lemma insert_before_not_strongly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and ptr and child and ref where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ insert_before ptr child ref →_h h'" and
    "¬ is_strongly_dom_component_safe ({ptr, cast child} ∪ (cast ‘ set_option ref)) {} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      e2 ← create_element document_ptr ''div'';
      return (e1, e2)
  }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "fst |?h0 ⊢ ?P|_r"
  let ?e2 = "snd |?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e1" and
          child="cast_element_ptr2node_ptr ?e2" and ref=None])
    by code_simp+
qed


lemma append_child_not_strongly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and ptr and child where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ append_child ptr child →_h h'" and
    "¬ is_strongly_dom_component_safe {ptr, cast child} {} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
```

```
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "do {
      document_ptr ← create_document;
      e1 ← create_element document_ptr ''div'';
      e2 ← create_element document_ptr ''div'';
      return (e1, e2)
  }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "fst |?h0 ⊢ ?P|_r"
  let ?e2 = "snd |?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e1" and child="cast_element_ptr2node_ptr
?e2"])
    by code_simp+
qed
```

2.1.13 get_owner_document

```
lemma get_owner_document_not_strongly_dom_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap" and
    h' and ptr and owner_document where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_owner_document ptr →_r owner_document →_h h'" and
    "¬ is_strongly_dom_component_safe {ptr} {cast owner_document} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder}) heap"
  let ?P = "do {
    doc ← create_document;
    create_element doc ''div''
  }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_element_ptr2object_ptr ?e1"])
    by code_simp+
qed


end
```

2.2 Shadow Root Components (Shadow_DOM_Components)

```
theory Shadow_DOM_Components
  imports
    Shadow_DOM.Shadow_DOM
    Core_DOM_Components
begin
```

2.2.1 get_component

```
global_interpretation l_get_component_Core_DOM_defs get_root_node get_root_node_locs to_tree_order
  defines get_component = a_get_component
    and is_strongly_dom_component_safe = a_is_strongly_dom_component_safe
```

```
    and is_weakly_dom_component_safe = a_is_weakly_dom_component_safe
  .

interpretation i_get_component?: l_get_component_Core_DOM
  heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name
  by(auto simp add: l_get_component_Core_DOM_def l_get_component_Core_DOM_axioms_def get_component_def
      is_strongly_dom_component_safe_def is_weakly_dom_component_safe_def instances)
declare l_get_component_Core_DOM_axioms [instances]
```

2.2.2 attach_shadow_root

```
lemma attach_shadow_root_not_strongly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'ShadowRoot::{equal,linorder}) heap" and
    h' and host and new_shadow_root_ptr where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ attach_shadow_root host m →_r new_shadow_root_ptr →_h h'" and
    "¬ is_strongly_dom_component_safe {cast host} {cast new_shadow_root_ptr} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'ShadowRoot::{equal,linorder}) heap"
  let ?P = "do {
    doc ← create_document;
    create_element doc ''div''
  }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and host="?e1"])
    by code_simp+
qed

locale l_get_dom_component_attach_shadow_root_Core_DOM =
  l_get_component_Core_DOM heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order
  get_parent get_parent_locs get_child_nodes get_child_nodes_locs get_dom_component
  is_strongly_component_safe is_weakly_component_safe get_root_node get_root_node_locs get_ancestors
  get_ancestors_locs get_disconnected_nodes get_disconnected_nodes_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name +
  l_attach_shadow_root_Shadow_DOM known_ptr set_shadow_root set_shadow_root_locs set_mode set_mode_locs
  attach_shadow_root type_wf get_tag_name get_tag_name_locs get_shadow_root get_shadow_root_locs +
  l_set_mode_Shadow_DOM type_wf set_mode set_mode_locs +
  l_set_shadow_root_Shadow_DOM type_wf set_shadow_root set_shadow_root_locs
  for known_ptr :: "('_::linorder) object_ptr ⇒ bool"
    and heap_is_wellformed :: "('_) heap ⇒ bool"
    and parent_child_rel :: "('_) heap ⇒ (('_) object_ptr × ('_) object_ptr) set"
    and type_wf :: "('_) heap ⇒ bool"
    and known_ptrs :: "('_) heap ⇒ bool"
    and to_tree_order :: "('_) object_ptr ⇒ (('_) heap, exception, ('_) object_ptr list) prog"
    and get_parent :: "('_) node_ptr ⇒ (('_) heap, exception, ('_) object_ptr option) prog"
    and get_parent_locs :: "(('_) heap ⇒ ('_) heap ⇒ bool) set"
    and get_child_nodes :: "('_) object_ptr ⇒ (('_) heap, exception, ('_) node_ptr list) prog"
    and get_child_nodes_locs :: "('_) object_ptr ⇒ (('_) heap ⇒ ('_) heap ⇒ bool) set"
```

```
    and get_dom_component :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_ancestors :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog"
    and get_ancestors_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_element_by_id ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and get_elements_by_class_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and get_elements_by_tag_name ::
    "(_) object_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr list) prog"
    and set_shadow_root ::
    "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ ((_) heap, exception, unit) prog"
    and set_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ ((_) heap, exception, unit) prog"
    and set_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, unit) prog set"
    and attach_shadow_root ::
    "(_) element_ptr ⇒ shadow_root_mode ⇒ ((_) heap, exception, (_) shadow_root_ptr) prog"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and is_strongly_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and is_weakly_component_safe ::
    "(_) object_ptr set ⇒ (_) object_ptr set ⇒ (_) heap ⇒ (_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin

lemma attach_shadow_root_is_weakly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ attach_shadow_root element_ptr shadow_root_mode →_h h'"
  assumes "ptr ≠ cast |h ⊢ attach_shadow_root element_ptr shadow_root_mode|_r"
  assumes "ptr ∉ set |h ⊢ get_dom_component (cast element_ptr)|_r"
  shows "preserved (get_M_Object ptr getter) h h'"
proof -
  obtain h2 h3 new_shadow_root_ptr where
    h2: "h ⊢ new_ShadowRoot_M →_h h2" and
    new_shadow_root_ptr: "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr" and
    h3: "h2 ⊢ set_mode new_shadow_root_ptr shadow_root_mode →_h h3" and
    h': "h3 ⊢ set_shadow_root element_ptr (Some new_shadow_root_ptr) →_h h'"
    using assms(4)
    by(auto simp add: attach_shadow_root_def elim!:  bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_tag_name_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_shadow_root_pure, rotated]
        split: if_splits)
  have "h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr"
    using new_shadow_root_ptr h2 h3 h'
    using assms(4)
    by(auto simp add: attach_shadow_root_def intro!: bind_returns_result_I
        bind_pure_returns_result_I[OF get_tag_name_pure] bind_pure_returns_result_I[OF get_shadow_root_pure]
        elim!:  bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_tag_name_pure, rotated]
        bind_returns_heap_E2[rotated, OF get_shadow_root_pure, rotated] split: if_splits)
  have "preserved (get_M_Object ptr getter) h h2"
    using h2 new_shadow_root_ptr
    by (metis (no_types, lifting)
        ‹h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr› assms(5)
        new_shadow_root_get_M_Object select_result_I2)

  have "ptr ≠ cast new_shadow_root_ptr"
    using ‹h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr› assms(5)
    by auto
```

```
  have "preserved (get_M_Object ptr getter) h2 h3"
    using set_mode_writes h3
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_mode_locs_def all_args_def)[1]
    using ‹ptr ≠ cast_shadow_root_ptr2object_ptr new_shadow_root_ptr›
    by (metis get_M_Mshadow_root_preserved3)

  have "element_ptr |∈| element_ptr_kinds h"
    using ‹h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr›
      attach_shadow_root_element_ptr_in_heap
    by blast
  have "ptr ≠ cast element_ptr"
    by (metis (no_types, lifting) ‹element_ptr |∈| element_ptr_kinds h› assms(1) assms(2) assms(3)
        assms(6) element_ptr_kinds_commutes is_OK_returns_result_E get_component_ok local.get_dom_component_ptr
        node_ptr_kinds_commutes select_result_I2)

  have "preserved (get_M_Object ptr getter) h3 h'"
    using set_shadow_root_writes h'
    apply(rule reads_writes_preserved2)
    apply(auto simp add: set_shadow_root_locs_def all_args_def)[1]
    by (metis ‹ptr ≠ cast_element_ptr2object_ptr element_ptr› get_M_Element_preserved8)

  show ?thesis
    using ‹preserved (get_M ptr getter) h h2› ‹preserved (get_M ptr getter) h2 h3›
      ‹preserved (get_M ptr getter) h3 h'›
    by(auto simp add: preserved_def)
qed
end

interpretation i_get_dom_component_attach_shadow_root?: l_get_dom_component_attach_shadow_root_Core_DOM
  known_ptr heap_is_wellformed parent_child_rel type_wf known_ptrs to_tree_order get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_component get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  set_shadow_root set_shadow_root_locs set_mode set_mode_locs attach_shadow_root get_disconnected_nodes
  get_disconnected_nodes_locs is_strongly_dom_component_safe is_weakly_dom_component_safe get_tag_name
  get_tag_name_locs get_shadow_root get_shadow_root_locs
  by(auto simp add: l_get_dom_component_attach_shadow_root_Core_DOM_def instances)
declare l_get_dom_component_attach_shadow_root_Core_DOM_axioms [instances]
```

2.2.3 get_shadow_root

```
lemma get_shadow_root_not_weakly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    element_ptr and shadow_root_ptr_opt and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_shadow_root element_ptr →_r shadow_root_ptr_opt →_h h'" and
    "¬ is_weakly_dom_component_safe {cast element_ptr} (cast ‘ set_option shadow_root_ptr_opt) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
```

```
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e1
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and element_ptr="?e1"])
    by code_simp+
qed

locale l_get_shadow_root_component_Shadow_DOM =
  l_get_shadow_root +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_component_Core_DOM +
  l_get_root_node_Core_DOM +
  l_get_root_node_wf_Core_DOM +
  l_remove_shadow_root_get_child_nodes
begin
lemma get_shadow_root_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
  shows "set |h ⊢ get_component (cast host)|_r ∩ set |h ⊢ get_component (cast shadow_root_ptr)|_r = {}"
proof -
  have "cast host |∈| object_ptr_kinds h"
    using assms(4) get_shadow_root_ptr_in_heap by auto
  then obtain host_c where host_c: "h ⊢ get_component (cast host) →_r host_c"
    by (meson  assms(1) assms(2) assms(3) get_component_ok is_OK_returns_result_E)
  obtain host_root where host_root: "h ⊢ get_root_node (cast host) →_r host_root"
    by (metis (no_types, lifting) bind_returns_heap_E get_component_def host_c is_OK_returns_result_I
        pure_def pure_eq_iff)

  have "cast shadow_root_ptr |∈| object_ptr_kinds h"
    using get_shadow_root_shadow_root_ptr_in_heap assms shadow_root_ptr_kinds_commutes
    using document_ptr_kinds_commutes by blast
  then obtain shadow_root_ptr_c where shadow_root_ptr_c:
    "h ⊢ get_component (cast shadow_root_ptr) →_r shadow_root_ptr_c"
    by (meson  assms(1) assms(2) assms(3) get_component_ok is_OK_returns_result_E)
  have "h ⊢ get_root_node (cast shadow_root_ptr) →_r cast shadow_root_ptr"
    using <cast shadow_root_ptr |∈| object_ptr_kinds h>
    by(auto simp add: get_root_node_def get_ancestors_def intro!: bind_pure_returns_result_I
        split: option.splits)

  have "host_root ≠ cast shadow_root_ptr"
  proof (rule ccontr, simp)
    assume "host_root = cast_shadow_root_ptr2object_ptr shadow_root_ptr"

    have "(cast shadow_root_ptr, host_root) ∈ (parent_child_rel h)*"
      using <host_root = cast_shadow_root_ptr2object_ptr shadow_root_ptr> by auto
    moreover have "(host_root, cast host) ∈ (parent_child_rel h)*"
      using get_root_node_parent_child_rel host_root assms
      by blast
    moreover have "(cast host, cast shadow_root_ptr) ∈ (a_host_shadow_root_rel h)"
      using assms(4) apply(auto simp add: a_host_shadow_root_rel_def)[1]
```

```
    by (metis (mono_tags, lifting) get_shadow_root_ptr_in_heap image_eqI is_OK_returns_result_I
        mem_Collect_eq prod.simps(2) select_result_I2)
  moreover have " acyclic (parent_child_rel h ∪ local.a_host_shadow_root_rel h)"
    using assms(1)[unfolded heap_is_wellformed_def]
    by auto
  ultimately show False
    using local.parent_child_rel_node_ptr
    by (metis (no_types, lifting) Un_iff ‹host_root = cast_shadow_root_ptr2object_ptr shadow_root_ptr›
        acyclic_def in_rtrancl_UnI rtrancl_into_trancl1)
qed
then have "host_c ≠ shadow_root_ptr_c"
  by (metis ‹h ⊢ get_root_node (cast_shadow_root_ptr2object_ptr shadow_root_ptr) →_r cast_shadow_root_ptr2object_ptr
shadow_root_ptr› assms(1) assms(2) assms(3) get_dom_component_ptr get_component_root_node_same
      host_c host_root local.get_root_node_parent_child_rel local.get_root_node_same_no_parent_parent_child_rel
      rtranclE shadow_root_ptr_c)
then have "set host_c ∩ set shadow_root_ptr_c = {}"
  using assms get_dom_component_no_overlap Shadow_DOM.a_heap_is_wellformed_def host_c
    shadow_root_ptr_c
  by blast
then show ?thesis
  using host_c shadow_root_ptr_c
  by auto
qed
end

interpretation i_get_shadow_root_component?: l_get_shadow_root_component_Shadow_DOM
  type_wf get_shadow_root get_shadow_root_locs get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_tag_name get_tag_name_locs known_ptr
  heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs
  get_disconnected_document get_disconnected_document_locs known_ptrs to_tree_order get_parent
  get_parent_locs get_component is_strongly_dom_component_safe is_weakly_dom_component_safe
  get_root_node get_root_node_locs get_ancestors get_ancestors_locs get_element_by_id
  get_elements_by_class_name get_elements_by_tag_name remove_shadow_root remove_shadow_root_locs
  by(auto simp add: l_get_shadow_root_component_Shadow_DOM_def instances)
declare l_get_shadow_root_component_Shadow_DOM_axioms [instances]
```

2.2.4 get_host

```
lemma get_host_not_weakly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    shadow_root_ptr and host and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_host shadow_root_ptr →_r host →_h h'" and
    "¬ is_weakly_dom_component_safe {cast shadow_root_ptr} {cast host} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
```

```
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return s1
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?s1 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and shadow_root_ptr="?s1"])
    by code_simp+
qed

locale l_get_host_component_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM +
  l_get_host +
  l_get_component_Core_DOM +
  l_get_shadow_root_component_Shadow_DOM
begin
lemma get_host_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_host shadow_root_ptr →_r host"
  shows "set |h ⊢ get_component (cast host)|_r ∩ set |h ⊢ get_component (cast shadow_root_ptr)|_r = {}"
proof -
  have "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
    using assms(1) assms(2) assms(4) local.shadow_root_host_dual by blast
  then show ?thesis
    using assms(1) assms(2) assms(3) local.get_shadow_root_is_component_unsafe by blast
qed
end

interpretation i_get_host_component?: l_get_host_component_Shadow_DOM
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs known_ptr type_wf heap_is_wellformed
  parent_child_rel heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs known_ptrs to_tree_order get_parent get_parent_locs get_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  remove_shadow_root remove_shadow_root_locs
  by(auto simp add: l_get_host_component_Shadow_DOM_def instances)
declare l_get_host_component_Shadow_DOM_axioms [instances]
```

2.2.5 get_root_node_si

```
locale l_get_component_get_root_node_si_Shadow_DOM =
  l_get_root_node_si_wf_Shadow_DOM +
  l_get_component_Core_DOM
begin

lemma get_root_node_si_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_root_node_si ptr' →_r root"
  shows "set |h ⊢ get_component ptr'|_r = set |h ⊢ get_component root|_r ∨
set |h ⊢ get_component ptr'|_r ∩ set |h ⊢ get_component root|_r = {}"
proof -
  have "ptr' |∈| object_ptr_kinds h"
    using get_ancestors_si_ptr_in_heap assms(4)
    by(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)
  then
  obtain c where "h ⊢ get_component ptr' →_r c"
```

```
    by (meson assms(1) assms(2) assms(3) local.get_component_ok select_result_I)
  moreover
  have "root |∈| object_ptr_kinds h"
    using get_ancestors_si_ptr assms(4)
    apply(auto simp add: get_root_node_si_def elim!: bind_returns_result_E2)[1]
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) empty_iff empty_set
        get_ancestors_si_ptrs_in_heap last_in_set)
  then
  obtain c' where "h ⊢ get_component root →_r c'"
    by (meson assms(1) assms(2) assms(3) local.get_component_ok select_result_I)
  ultimately show ?thesis
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) local.get_dom_component_no_overlap
        select_result_I2)
qed
end

interpretation i_get_component_get_root_node_si?: l_get_component_get_root_node_si_Shadow_DOM
  type_wf known_ptr known_ptrs get_parent get_parent_locs get_child_nodes get_child_nodes_locs
  get_host get_host_locs get_ancestors_si get_ancestors_si_locs get_root_node_si get_root_node_si_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs get_tag_name
  get_tag_name_locs heap_is_wellformed parent_child_rel heap_is_wellformed_Core_DOM get_disconnected_document
  get_disconnected_document_locs to_tree_order get_component is_strongly_dom_component_safe
  is_weakly_dom_component_safe get_root_node get_root_node_locs get_ancestors get_ancestors_locs
  get_element_by_id get_elements_by_class_name get_elements_by_tag_name
  by(auto simp add: l_get_component_get_root_node_si_Shadow_DOM_def instances)
declare l_get_component_get_root_node_si_Shadow_DOM_axioms [instances]
```

2.2.6 get_assigned_nodes

```
lemma assigned_nodes_not_weakly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    node_ptr and nodes and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ assigned_nodes node_ptr →_r nodes →_h h'" and
    "¬ is_weakly_dom_component_safe {cast node_ptr} (cast ` set nodes) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Closed;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e3
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
```

```
  let ?e3 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and node_ptr="?e3"])
    by code_simp+
qed

lemma get_composed_root_node_not_weakly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    ptr and root and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ get_root_node_si ptr →_r root →_h h'" and
    "¬ is_weakly_dom_component_safe {ptr} {root} h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Closed;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e3
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e3 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and ptr="cast_{element_ptr2object_ptr} ?e3"])
    by code_simp+
qed

lemma assigned_slot_not_weakly_component_safe:
  obtains
    h :: "('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder}, 'element_ptr::{equal,linorder},
'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder}, 'shadow_root_ptr::{equal,linorder},
'Object::{equal,linorder}, 'Node::{equal,linorder}, 'Element::{equal,linorder},
'CharacterData::{equal,linorder}, 'Document::{equal,linorder}, 'Shadowroot::{equal,linorder}) heap" and
    node_ptr and slot_opt and h' where
    "heap_is_wellformed h" and "type_wf h" and "known_ptrs h" and
    "h ⊢ assigned_slot node_ptr →_r slot_opt →_h h'" and
    "¬ is_weakly_dom_component_safe {cast node_ptr} (cast ‘ set_option slot_opt) h h'"
proof -
  let ?h0 = "Heap fmempty ::('object_ptr::{equal,linorder}, 'node_ptr::{equal,linorder},
'element_ptr::{equal,linorder}, 'character_data_ptr::{equal,linorder}, 'document_ptr::{equal,linorder},
'shadow_root_ptr::{equal,linorder}, 'Object::{equal,linorder}, 'Node::{equal,linorder},
'Element::{equal,linorder}, 'CharacterData::{equal,linorder}, 'Document::{equal,linorder},
```

```
'Shadowroot::{equal,linorder}) heap"
  let ?P = "do {
        document_ptr ← create_document;
        html ← create_element document_ptr ''html'';
        append_child (cast document_ptr) (cast html);
        head ← create_element document_ptr ''head'';
        append_child (cast html) (cast head);
        body ← create_element document_ptr ''body'';
        append_child (cast html) (cast body);
        e1 ← create_element document_ptr ''div'';
        append_child (cast body) (cast e1);
        e2 ← create_element document_ptr ''div'';
        append_child (cast e1) (cast e2);
        s1 ← attach_shadow_root e1 Open;
        e3 ← create_element document_ptr ''slot'';
        append_child (cast s1) (cast e3);
        return e2
      }"
  let ?h1 = "|?h0 ⊢ ?P|_h"
  let ?e2 = "|?h0 ⊢ ?P|_r"

  show thesis
    apply(rule that[where h="?h1" and node_ptr="cast_element_ptr2node_ptr ?e2"])
    by code_simp+
qed

locale l_assigned_nodes_component_Shadow_DOM =
  l_get_tag_name +
  l_get_child_nodes +
  l_heap_is_wellformed_Shadow_DOM +
  l_find_slot_Shadow_DOM +
  l_assigned_nodes_Shadow_DOM +
  l_assigned_nodes_wf_Shadow_DOM +
  l_get_component_Core_DOM +
  l_adopt_node_Core_DOM +
  l_remove_child_Core_DOM +
  l_remove_child_wf2 +
  l_insert_before_wf +
  l_insert_before_wf2 +
  l_insert_before_Core_DOM _ _ _ _ _ _ get_ancestors_si get_ancestors_si_locs +
  l_append_child_Core_DOM +
  l_append_child_wf_Core_DOM _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ get_ancestors_si get_ancestors_si_locs
+
  l_set_disconnected_nodes_get_tag_name +
  l_set_shadow_root_get_child_nodes +
  l_set_child_nodes_get_tag_name +
  l_get_shadow_root_component_Shadow_DOM +
  l_remove_shadow_root_Shadow_DOM +
  l_remove_shadow_root_get_tag_name +
  l_set_disconnected_nodes_get_shadow_root +
  l_set_child_nodes_get_shadow_root +
  l_remove_shadow_root_wf_Shadow_DOM
begin
lemma find_slot_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ find_slot open_flag node_ptr →_r Some slot"
  shows "set |h ⊢ get_component (cast node_ptr)|_r ∩ set |h ⊢ get_component (cast slot)|_r = {}"
proof -
  obtain host shadow_root_ptr to where
    "h ⊢ get_parent node_ptr →_r Some (cast host)" and
    "h ⊢ get_shadow_root host →_r Some shadow_root_ptr" and
    "h ⊢ to_tree_order (cast shadow_root_ptr) →_r to" and
    "cast slot ∈ set to"
```

```
    using assms(4)
    apply(auto simp add: find_slot_def first_in_tree_order_def elim!: bind_returns_result_E2
        map_filter_M_pure_E[where y=slot]
        split: option.splits if_splits list.splits intro!: map_filter_M_pure bind_pure_I)[1]
    by (metis element_ptr_casts_commute3)+

  have "node_ptr |∈| node_ptr_kinds h"
    using assms(4) find_slot_ptr_in_heap by blast
  then obtain node_ptr_c where node_ptr_c: "h ⊢ get_component (cast node_ptr) →r node_ptr_c"
    using assms(1) assms(2) assms(3) get_component_ok is_OK_returns_result_E
      node_ptr_kinds_commutes[symmetric]
    by metis

  then have "cast host ∈ set node_ptr_c"
    using ‹h ⊢ get_parent node_ptr →r Some (cast host)› get_component_parent_inside assms(1)
      assms(2) assms(3) get_dom_component_ptr
    by blast

  then have "h ⊢ get_component (cast host) →r node_ptr_c"
    using ‹h ⊢ get_parent node_ptr →r Some (cast host)› get_component_subset a_heap_is_wellformed_def
      assms(1) assms(2) assms(3) node_ptr_c
    by blast

  moreover have "slot |∈| element_ptr_kinds h"
    using assms(4) find_slot_slot_in_heap by blast
  then obtain slot_c where slot_c: "h ⊢ get_component (cast slot) →r slot_c"
    using a_heap_is_wellformed_def assms(1) assms(2) assms(3) get_component_ok is_OK_returns_result_E
      node_ptr_kinds_commutes[symmetric] element_ptr_kinds_commutes[symmetric]
    by metis
  then have "cast shadow_root_ptr ∈ set slot_c"
    using ‹h ⊢ to_tree_order (cast shadow_root_ptr) →r to› ‹cast slot ∈ set to›
      get_component_to_tree_order assms(1) assms(2) assms(3) get_dom_component_ptr
    by blast
  then have "h ⊢ get_component (cast shadow_root_ptr) →r slot_c"
    using  ‹h ⊢ get_shadow_root host →r Some shadow_root_ptr› get_component_subset assms(1) assms(2)
      assms(3) slot_c
    by blast

  ultimately show ?thesis
    using get_shadow_root_is_component_unsafe assms ‹h ⊢ get_shadow_root host →r Some shadow_root_ptr›
      node_ptr_c slot_c
    by fastforce
qed

lemma assigned_slot_pure:
  "pure (assigned_slot node_ptr) h"
  apply(auto simp add: assigned_slot_def a_find_slot_def intro!: bind_pure_I  split: option.splits)[1]
  by(auto simp add: first_in_tree_order_def intro!: bind_pure_I map_filter_M_pure split: list.splits)

lemma assigned_slot_is_strongly_dom_component_safe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ assigned_slot node_ptr →r slot_opt →h h'"
  assumes "∀shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h). h ⊢ get_mode shadow_root_ptr →r Closed"
  shows "is_strongly_dom_component_safe {cast node_ptr} (cast ‘ set_option slot_opt) h h'"
proof -
  have "h = h'"
    using assms(5) assigned_slot_pure
    by (meson assms(4) pure_returns_heap_eq returns_result_heap_def)
  obtain parent_opt where parent_opt: "h ⊢ get_parent node_ptr →r parent_opt"
    using assms(4)
    by (auto simp add: assigned_slot_def a_find_slot_def returns_result_heap_def
        elim!: bind_returns_result_E2 split: option.splits split: if_splits)
  then
```

```
have "slot_opt = None"
proof (induct parent_opt)
  case None
  then show ?case
    using assms(4)
    apply(auto simp add: assigned_slot_def a_find_slot_def returns_result_heap_def
        elim!: bind_returns_result_E2 split: option.splits split: if_splits)[1]
    by (meson option.distinct(1) returns_result_eq)+
next
  case (Some parent)
  then show ?case
  proof (cases "is_element_ptr_kind parent")
    case True
    then obtain host where host: "parent = cast_element_ptr2object_ptr host"
      by (metis (no_types, lifting) case_optionE element_ptr_casts_commute3 node_ptr_casts_commute)
    moreover have "host |∈| element_ptr_kinds h"
      using Some.prems element_ptr_kinds_commutes host local.get_parent_parent_in_heap
        node_ptr_kinds_commutes
      by blast
    ultimately obtain shadow_root_opt where shadow_root_opt:
      "h ⊢ get_shadow_root host →_r shadow_root_opt"
      using get_shadow_root_ok
      by (meson assms(2) is_OK_returns_result_E)
    then show ?thesis
    proof (induct shadow_root_opt)
      case None
      then show ?case
        using assms(4)
        apply(auto dest!: returns_result_eq[OF ‹h ⊢ get_parent node_ptr →_r Some parent›]
            simp add: assigned_slot_def a_find_slot_def returns_result_heap_def elim!: bind_returns_result_E2
            split: option.splits split: if_splits)[1]
        using host select_result_I2 by fastforce+
    next
      case (Some shadow_root_ptr)
      have "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
        using Some.prems assms(1) local.get_shadow_root_shadow_root_ptr_in_heap by blast
      then
      have "h ⊢ get_mode shadow_root_ptr →_r Closed"
        using assms by simp
      have "¬h ⊢ get_mode shadow_root_ptr →_r Open"
      proof (rule ccontr, simp)
        assume "h ⊢ get_mode shadow_root_ptr →_r Open"
        then have "Open = Closed"
          using returns_result_eq ‹h ⊢ get_mode shadow_root_ptr →_r Closed›
          by fastforce
        then show False
          by simp
      qed
      then show ?case
        using assms(4) Some parent_opt host
        by(auto dest!: returns_result_eq[OF ‹h ⊢ get_parent node_ptr →_r Some parent›]
            returns_result_eq[OF ‹h ⊢ get_shadow_root host →_r Some shadow_root_ptr›]
            simp add: assigned_slot_def a_find_slot_def returns_result_heap_def
            elim!: bind_returns_result_E2 split: option.splits split: if_splits)
    qed
  next
    case False
    then show ?thesis
      using assms(4)
      by(auto dest!: returns_result_eq[OF ‹h ⊢ get_parent node_ptr →_r Some parent›]
          simp add: assigned_slot_def a_find_slot_def returns_result_heap_def
          elim!: bind_returns_result_E2 split: option.splits split: if_splits)
  qed
```

```
  qed
  then show ?thesis
    using ‹h = h'›
    by(auto simp add: is_strongly_dom_component_safe_def preserved_def)
qed

lemma assigned_nodes_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ assigned_nodes element_ptr →r nodes"
  assumes "node_ptr ∈ set nodes"
  shows "set |h ⊢ get_component (cast element_ptr)|r ∩ set |h ⊢ get_component (cast node_ptr)|r = {}"
proof -
  have "h ⊢ find_slot False node_ptr →r Some element_ptr"
    using assms(4) assms(5)
    by(auto simp add: assigned_nodes_def elim!: bind_returns_result_E2
        dest!: filter_M_holds_for_result[where x=node_ptr] intro!: bind_pure_I split: if_splits)
  then show ?thesis
    using assms find_slot_is_component_unsafe
    by blast
qed

lemma flatten_dom_assigned_nodes_become_children:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ flatten_dom →h h'"
  assumes "h ⊢ assigned_nodes slot →r nodes"
  assumes "nodes ≠ []"
  shows "h' ⊢ get_child_nodes (cast slot) →r nodes"
proof -
  obtain tups h2 element_ptrs shadow_root_ptrs where
    "h ⊢ element_ptr_kinds_M →r element_ptrs" and
    tups: "h ⊢ map_filter_M2 (λelement_ptr. do {
      tag ← get_tag_name element_ptr;
      assigned_nodes ← assigned_nodes element_ptr;
      (if tag = ''slot'' ∧ assigned_nodes ≠ [] then return (Some (element_ptr, assigned_nodes))
else return None)}) element_ptrs →r tups" (is "h ⊢ map_filter_M2 ?f element_ptrs →r tups") and
    h2: "h ⊢ forall_M (λ(slot, assigned_nodes). do {
      get_child_nodes (cast slot) ⤜ forall_M remove;
      forall_M (append_child (cast slot)) assigned_nodes
    }) tups →h h2" and
    "h2 ⊢ shadow_root_ptr_kinds_M →r shadow_root_ptrs" and
    h': "h2 ⊢ forall_M (λshadow_root_ptr. do {
      host ← get_host shadow_root_ptr;
      get_child_nodes (cast host) ⤜ forall_M remove;
      get_child_nodes (cast shadow_root_ptr) ⤜ forall_M (append_child (cast host));
      remove_shadow_root host
    }) shadow_root_ptrs →h h'"
    using ‹h ⊢ flatten_dom →h h'›
    apply(auto simp add: flatten_dom_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF ElementMonad.ptr_kinds_M_pure, rotated]
        bind_returns_heap_E2[rotated, OF ShadowRootMonad.ptr_kinds_M_pure, rotated])[1]
    apply(drule pure_returns_heap_eq)
    by(auto intro!: map_filter_M2_pure bind_pure_I)

  have all_tups_slot: "⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set tups ⟹
h ⊢ get_tag_name slot →r ''slot''"
    using tups
    apply(induct element_ptrs arbitrary: tups)
    by(auto elim!: bind_returns_result_E2 split: if_splits intro!: map_filter_M2_pure bind_pure_I)

  have "distinct element_ptrs"
    using ‹h ⊢ element_ptr_kinds_M →r element_ptrs›
    by auto
  then
```

```
  have "distinct tups"
    using tups
    apply(induct element_ptrs arbitrary: tups)
    by(auto elim!: bind_returns_result_E2 intro!: map_filter_M2_pure bind_pure_I
        split: option.splits if_splits intro: map_filter_pure_foo[rotated] )


  have "slot ∈ set element_ptrs"
    using assms(5) assigned_nodes_ptr_in_heap ‹h ⊢ element_ptr_kinds_M →r element_ptrs›
    by auto
  then
  have "(slot, nodes) ∈ set tups"
    apply(rule map_filter_M2_in_result[OF tups])
    apply(auto intro!: bind_pure_I)[1]
    apply(intro bind_pure_returns_result_I)
    using assms assigned_nodes_slot_is_slot
    by(auto intro!: bind_pure_returns_result_I)

  have "⋀slot nodes. (slot, nodes) ∈ set tups ⟹ h ⊢ assigned_nodes slot →r nodes"
    using tups
    apply(induct element_ptrs arbitrary: tups)
    by(auto elim!: bind_returns_result_E2 intro!: map_filter_M2_pure bind_pure_I split: if_splits)
  then
  have elementwise_eq: "⋀slot slot' nodes nodes'. (slot, nodes) ∈ set tups ⟹
(slot', nodes') ∈ set tups ⟹ slot = slot' ⟹ nodes = nodes'"
    by (meson returns_result_eq)

  have "⋀slot nodes. (slot, nodes) ∈ set tups ⟹ distinct nodes"
    using ‹⋀slot nodes. (slot, nodes) ∈ set tups ⟹ h ⊢ assigned_nodes slot →r nodes›
      assigned_nodes_distinct
    using assms(1) by blast

  have "⋀slot slot' nodes nodes'. (slot, nodes) ∈ set tups ⟹ (slot', nodes') ∈ set tups ⟹
slot ≠ slot' ⟹ set nodes ∩ set nodes' = {}"
    using ‹⋀slot nodes. (slot, nodes) ∈ set tups ⟹ h ⊢ assigned_nodes slot →r nodes›
      assigned_nodes_different_ptr assms(1) assms(2) assms(3) by blast

  have "h ⊢ get_tag_name slot →r ''slot''"
    using ‹(slot, nodes) ∈ set tups› all_tups_slot by blast
  then have "h2 ⊢ get_tag_name slot →r ''slot''"
    using h2
  proof(induct tups arbitrary: h, simp)
    case (Cons x xs)
    obtain xc ha hb slot' nodes' where
      "x = (slot', nodes')" and
      "h ⊢ get_child_nodes (cast_element_ptr2object_ptr slot') →r xc" and
      ha: "h ⊢ forall_M remove xc →h ha" and
      hb: "ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →h hb" and
      remainder: "hb ⊢ forall_M (λ(slot, assigned_nodes). Heap_Error_Monad.bind
(get_child_nodes (cast_element_ptr2object_ptr slot)) (λx. Heap_Error_Monad.bind (forall_M remove x)
(λ_. forall_M (append_child (cast_element_ptr2object_ptr slot)) assigned_nodes))) xs →h h2"
      using Cons(3)
      by (auto elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated] bind_returns_result_E
          bind_returns_result_E2[rotated, OF get_child_nodes_pure, rotated] split: prod.splits)

    have "ha ⊢ get_tag_name slot →r ''slot''"
      using ‹h ⊢ get_tag_name slot →r ''slot''› ha
    proof(induct xc arbitrary: h, simp)
      case (Cons a yc)
      obtain hb1 where
        hb1: "h ⊢ remove a →h hb1" and
        hba: "hb1 ⊢ forall_M remove yc →h ha"
```

```
      using Cons
      by (auto elim!: bind_returns_heap_E)
    have "hb1 ⊢ get_tag_name slot →r ''slot''"
      using ‹h ⊢ get_tag_name slot →r ''slot''› hb1
      by(auto simp add: remove_child_locs_def set_child_nodes_get_tag_name
          set_disconnected_nodes_get_tag_name
          dest!: reads_writes_separate_forwards[OF get_tag_name_reads remove_writes])
    then show ?case
      using hba Cons(1) by simp
  qed
  then
  have "hb ⊢ get_tag_name slot →r ''slot''"
    using hb
  proof (induct nodes' arbitrary: ha, simp)
    case (Cons a nodes')
    obtain ha1 where
      ha1: "ha ⊢ append_child (cast slot') a →h ha1" and
      hb: "ha1 ⊢ forall_M (append_child (cast slot')) nodes' →h hb"
      using Cons
      by (auto elim!: bind_returns_heap_E)
    have "ha1 ⊢ get_tag_name slot →r ''slot''"
      using ‹ha ⊢ get_tag_name slot →r ''slot''› ha1
      by(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def
          adopt_node_locs_def remove_child_locs_def set_child_nodes_get_tag_name
          set_disconnected_nodes_get_tag_name
          dest!: reads_writes_separate_forwards[OF get_tag_name_reads insert_before_writes]
          split: if_splits)
    then show ?case
      using ha1 hb Cons(1)
      by simp
  qed
  then
  show ?case
    using Cons(1) remainder by simp
qed

  have "h2 ⊢ get_child_nodes (cast slot) →r nodes ∧ heap_is_wellformed h2 ∧ type_wf h2 ∧ known_ptrs h2"
    using ‹(slot, nodes) ∈ set tups›
    using h2 assms(1) assms(2) assms(3) ‹distinct tups› all_tups_slot elementwise_eq
    using ‹⋀slot slot' assigned_nodes nodes'. (slot, assigned_nodes) ∈ set tups ⟹
(slot', nodes') ∈ set tups ⟹ slot ≠ slot' ⟹ set assigned_nodes ∩ set nodes' = {}›
    using ‹⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set tups ⟹ distinct assigned_nodes›
  proof(induct tups arbitrary: h, simp)
    case (Cons x xs)
    obtain xc ha hb slot' nodes' where
      "x = (slot', nodes')" and
      "h ⊢ get_child_nodes (cast_element_ptr2object_ptr slot') →r xc" and
      ha: "h ⊢ forall_M remove xc →h ha" and
      hb: "ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →h hb" and
      remainder: "hb ⊢ forall_M (λ(slot, assigned_nodes). Heap_Error_Monad.bind
(get_child_nodes (cast_element_ptr2object_ptr slot)) (λx. Heap_Error_Monad.bind (forall_M remove x)
(λ_. forall_M (append_child (cast_element_ptr2object_ptr slot)) assigned_nodes))) xs →h h2"
      using Cons(3)
      by (auto elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated] bind_returns_result_E
          bind_returns_result_E2[rotated, OF get_child_nodes_pure, rotated] split: prod.splits)

    have "⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set xs ⟹ h ⊢ get_tag_name slot →r ''slot''"
      using Cons by auto

    have "heap_is_wellformed ha" and "type_wf ha" and "known_ptrs ha"
      using Cons(4) Cons(5) Cons(6) ‹h ⊢ forall_M remove xc →h ha›
      apply(induct xc arbitrary: h)
```

```
    apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        simp add: remove_def split: option.splits)[1]
    apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        simp add: remove_def split: option.splits)[1]
    apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        simp add: remove_def split: option.splits)[1]
    apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        simp add: remove_def split: option.splits)[1]
    apply (meson local.remove_child_heap_is_wellformed_preserved local.remove_child_preserves_known_ptrs
        local.remove_child_preserves_type_wf)
    apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        simp add: remove_def split: option.splits)[1]
    apply (meson local.remove_child_heap_is_wellformed_preserved local.remove_child_preserves_known_ptrs
        local.remove_child_preserves_type_wf)
    apply(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_parent_pure, rotated]
        simp add: remove_def split: option.splits)[1]
    using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
      remove_child_preserves_known_ptrs apply metis
    done
  then
  have "heap_is_wellformed hb" and "type_wf hb" and "known_ptrs hb"
    using ‹ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →_h hb›
    apply(induct nodes' arbitrary: ha)
    apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
    apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
    apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
    apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
    apply (metis insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
        insert_before_preserves_known_ptrs)
    apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
    apply (metis insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
        insert_before_preserves_known_ptrs)
    apply(auto elim!: bind_returns_heap_E simp add: append_child_def)[1]
    apply (metis insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
        insert_before_preserves_known_ptrs)
    done

  {
    fix slot assigned_nodes
    assume "(slot, assigned_nodes) ∈ set xs"
    then have "h ⊢ get_tag_name slot →_r ''slot''"
      using ‹⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set xs ⟹
h ⊢ get_tag_name slot →_r ''slot''›
      by auto
    then have "ha ⊢ get_tag_name slot →_r ''slot''"
      using ‹h ⊢ forall_M remove xc →_h ha›
      apply(induct xc arbitrary: h)
      by(auto simp add: remove_child_locs_def set_child_nodes_get_tag_name
          set_disconnected_nodes_get_tag_name
          dest!: reads_writes_separate_forwards[OF get_tag_name_reads remove_writes]
          elim!: bind_returns_heap_E)
    then have "hb ⊢ get_tag_name slot →_r ''slot''"
      using ‹ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →_h hb›
      apply(induct nodes' arbitrary: ha)
      by(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def adopt_node_locs_def
          remove_child_locs_def set_child_nodes_get_tag_name set_disconnected_nodes_get_tag_name
          dest!: reads_writes_separate_forwards[OF get_tag_name_reads insert_before_writes]
          elim!: bind_returns_heap_E
          split: if_splits)
  } note tag_names_same = this

  show ?case
  proof(cases "slot' = slot")
```

```
      case True
      then
      have "nodes' = nodes"
        using Cons.prems(1) Cons.prems(8) ‹x = (slot', nodes')›
        by (metis list.set_intros(1))
      then
      have "(slot, nodes) ∉ set xs"
        using Cons.prems(6) True ‹x = (slot', nodes')› by auto

      have "ha ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r []"
        using remove_for_all_empty_children Cons.prems(3) Cons.prems(4) Cons.prems(5) True
          ‹h ⊢ forall_M remove xc →_h ha›
        using ‹h ⊢ get_child_nodes (cast_element_ptr2object_ptr slot') →_r xc›
        by blast
      then
      have "hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes"
        using  append_child_for_all_on_no_children[OF ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs
hb›]
          True ‹nodes' = nodes›
        using ‹ha ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot')) nodes' →_h hb›
        using ‹(slot, nodes) ∈ set tups› ‹⋀slot assigned_nodes. (slot, assigned_nodes) ∈ set tups ⟹
distinct assigned_nodes› ‹heap_is_wellformed ha› ‹known_ptrs ha› ‹type_wf ha›
          local.append_child_for_all_on_no_children
        by blast
      with ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
      show ?thesis
        using ‹(slot, nodes) ∉ set xs› remainder
        using ‹⋀slot slot' assigned_nodes nodes'. (slot, assigned_nodes) ∈ set (x#xs) ⟹
(slot', nodes') ∈ set (x#xs) ⟹ slot = slot' ⟹ assigned_nodes = nodes'›
        using ‹(slot, nodes) ∈ set (x # xs)›
        using ‹⋀slot slot' assigned_nodes nodes'. (slot, assigned_nodes) ∈ set (x#xs) ⟹
(slot', nodes') ∈ set (x#xs) ⟹ slot ≠ slot' ⟹ set assigned_nodes ∩ set nodes' = {}›
      proof(induct xs arbitrary: hb, simp)
        case (Cons y ys)
        obtain yc hba hbb slot'' nodes'' where
          "y = (slot'', nodes'')" and
          "hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →_r yc" and
          "hb ⊢ forall_M remove yc →_h hba" and
          "hba ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot'')) nodes'' →_h hbb" and
          remainder: "hbb ⊢ forall_M (λ(slot, assigned_nodes). Heap_Error_Monad.bind
(get_child_nodes (cast_element_ptr2object_ptr slot)) (λx. Heap_Error_Monad.bind (forall_M remove x)
(λ_. forall_M (append_child (cast_element_ptr2object_ptr slot)) assigned_nodes))) ys →_h h2"
          using Cons(7)
          by (auto elim!: bind_returns_heap_E
              bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated] split: prod.splits)

        have "slot ≠ slot''"
          by (metis Cons.prems(5) Cons.prems(7) Cons.prems(8) ‹y = (slot'', nodes'')›
              list.set_intros(1) list.set_intros(2))
        then have "set nodes ∩ set nodes'' = {}"
          by (metis Cons.prems(8) Cons.prems(9) ‹y = (slot'', nodes'')› list.set_intros(1)
              list.set_intros(2))

        have "hba ⊢ get_child_nodes (cast slot) →_r nodes ∧
heap_is_wellformed hba ∧ type_wf hba ∧ known_ptrs hba"
          using ‹hb ⊢ get_child_nodes (cast slot) →_r nodes›
          using ‹hb ⊢ forall_M remove yc →_h hba›
          using ‹hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →_r yc›
          using ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs hb›
        proof(induct yc arbitrary: hb, simp)
          case (Cons a yc)
          obtain hb1 where
            hb1: "hb ⊢ remove a →_h hb1" and
```

```
      hba: "hb1 ⊢ forall_M remove yc →_h hba"
      using Cons
      by (auto elim!: bind_returns_heap_E)
    have "hb ⊢ get_parent a →_r Some (cast slot'')"
      using Cons.prems(3) Cons.prems(4) Cons.prems(5) Cons.prems(6) local.child_parent_dual
      by auto

    moreover
    have "heap_is_wellformed hb1" and "type_wf hb1" and "known_ptrs hb1"
      using ‹hb ⊢ remove a →_h hb1›
      apply(auto simp add: remove_def elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
      using ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
        remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
        remove_child_preserves_known_ptrs
      apply blast
      using ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
        remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
        remove_child_preserves_known_ptrs
      apply blast
      using ‹hb ⊢ remove a →_h hb1›
      apply(auto simp add: remove_def elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
      using ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
        remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
        remove_child_preserves_known_ptrs
      apply blast
      using ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
        remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
        remove_child_preserves_known_ptrs
      apply blast
      using ‹hb ⊢ remove a →_h hb1›
      apply(auto simp add: remove_def elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
      using ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
        remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
        remove_child_preserves_known_ptrs
      apply blast
      using ‹heap_is_wellformed hb› and ‹type_wf hb› and ‹known_ptrs hb›
        remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
        remove_child_preserves_known_ptrs
      apply blast
      done
    moreover have "hb1 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →_r yc"
      using ‹hb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot'') →_r a # yc› hb1
      using remove_removes_child ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs hb›
      by simp
    moreover have "hb1 ⊢ get_child_nodes (cast slot) →_r nodes"
      using Cons(2) hb1 set_child_nodes_get_child_nodes_different_pointers
        ‹hb ⊢ get_parent a →_r Some (cast slot'')› ‹slot ≠ slot''›
      apply(auto simp add: remove_child_locs_def elim!: bind_returns_heap_E
          dest!: reads_writes_separate_forwards[OF get_child_nodes_reads remove_writes])[1]
      by (metis cast_element_ptr2node_ptr_inject cast_node_ptr2object_ptr_inject)
    ultimately show ?thesis
      using ‹hb1 ⊢ forall_M remove (yc) →_h hba› Cons
      by auto
  qed


  then have "hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb"
    using ‹hba ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot'')) nodes'' →_h hbb›
    using ‹set nodes ∩ set nodes'' = {}›
```

```
proof(induct nodes'' arbitrary: hba, simp)
  case (Cons a nodes'')
  then have "hba ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes" and
    "heap_is_wellformed hba" and
    "type_wf hba" and
    "known_ptrs hba"
    by auto
  obtain hba1 where
    hba1: "hba ⊢ append_child (cast slot'') a →h hba1" and
    "hba1 ⊢ forall_M (append_child (cast slot'')) nodes'' →h hbb"
    using Cons(3)
    by (auto elim!: bind_returns_heap_E)

  have "heap_is_wellformed hba1" and "type_wf hba1" and "known_ptrs hba1"
    using ‹hba ⊢ append_child (cast slot'') a →h hba1›
    apply(auto simp add: append_child_def)[1]
    using ‹heap_is_wellformed hba› and ‹type_wf hba› and ‹known_ptrs hba›
      insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
      insert_before_preserves_known_ptrs
    apply blast
    using ‹hba ⊢ append_child (cast slot'') a →h hba1›
    apply(auto simp add: append_child_def)[1]
    using ‹heap_is_wellformed hba› and ‹type_wf hba› and ‹known_ptrs hba›
      insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
      insert_before_preserves_known_ptrs
    apply blast
    using ‹hba ⊢ append_child (cast slot'') a →h hba1›
    apply(auto simp add: append_child_def)[1]
    using ‹heap_is_wellformed hba› and ‹type_wf hba› and ‹known_ptrs hba›
      insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
      insert_before_preserves_known_ptrs
    apply blast
    done

  moreover
  have "a ∉ set nodes"
    using ‹set nodes ∩ set (a # nodes'') = {}›
    by auto


  moreover
  obtain parent_opt where "hba ⊢ get_parent a →r parent_opt"
    using insert_before_child_in_heap hba1 get_parent_ok unfolding append_child_def
    by (meson Cons.prems(1) is_OK_returns_heap_I is_OK_returns_result_E)
  then
  have "hba1 ⊢ get_child_nodes (cast slot) →r nodes"
  proof (induct parent_opt)
    case None
    then show ?case
      using ‹hba ⊢ append_child (cast slot'') a →h hba1›
      using ‹hba ⊢ get_child_nodes (cast slot) →r nodes›
      using ‹slot ≠ slot''›
      apply(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def
          adopt_node_locs_def remove_child_locs_def
          elim!: reads_writes_separate_forwards[OF get_child_nodes_reads insert_before_writes])[1]
      by (metis cast_element_ptr2node_ptr_inject cast_node_ptr2object_ptr_inject
          set_child_nodes_get_child_nodes_different_pointers)
  next
    case (Some parent)
    have "parent ≠ cast slot"
      apply(rule ccontr, simp)
      using Cons(2)
      apply -
```

```
            apply(rule get_parent_child_dual[OF ‹hba ⊢ get_parent a →r Some parent›])
            apply(auto)[1]
            using ‹a ∉ set nodes› returns_result_eq
            by fastforce
          show ?case
            apply(rule reads_writes_separate_forwards)
            apply(fact get_child_nodes_reads)
            apply(fact insert_before_writes)
            apply(fact ‹hba ⊢ append_child (cast slot'') a →h hba1›[unfolded append_child_def])
            apply(fact ‹hba ⊢ get_child_nodes (cast slot) →r nodes›)
            using ‹hba ⊢ get_parent a →r Some parent› ‹parent ≠ cast slot› ‹slot ≠ slot''›
            apply(auto simp add: insert_before_locs_def adopt_node_locs_def adopt_node_locs_def
                remove_child_locs_def)[1]
            apply (simp_all add: set_child_nodes_get_child_nodes_different_pointers
                set_disconnected_nodes_get_child_nodes)
            by (metis cast_element_ptr2node_ptr_inject cast_node_ptr2object_ptr_inject
                set_child_nodes_get_child_nodes_different_pointers)
        qed
        moreover
        have "set nodes ∩ set nodes'' = {}"
          using Cons.prems(3) by auto
        ultimately show ?case
          using Cons.hyps  ‹hba1 ⊢ forall_M (append_child (cast_element_ptr2object_ptr slot'')) nodes'' →h
hbb›
          by blast
      qed
      show ?case
        apply(rule Cons(1))
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using ‹hbb ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes ∧
heap_is_wellformed hbb ∧ type_wf hbb ∧ known_ptrs hbb›
        apply(auto)[1]
        using Cons.prems(5) apply auto[1]
        apply (simp add: remainder)
        using Cons.prems(7) apply auto[1]
        apply (simp add: True ‹nodes' = nodes› ‹x = (slot', nodes')›)
        by (metis Cons.prems(9) insert_iff list.simps(15))
    qed
  next
    case False
    then have "nodes' ≠ nodes"
      using Cons.prems(1) Cons.prems(9) ‹x = (slot', nodes')›
      by (metis assms(6) inf.idem list.set_intros(1) set_empty2)
    then
    have "(slot, nodes) ∈ set xs"
      using Cons.prems(1) ‹x = (slot', nodes')›
      by auto
    then show ?thesis
      using Cons(1)[simplified, OF ‹(slot, nodes) ∈ set xs› remainder
          ‹heap_is_wellformed hb› ‹type_wf hb› ‹known_ptrs hb›]
      using Cons.prems(6) tag_names_same  Cons.prems(8) Cons.prems(9)
      by (smt Cons.prems(10) distinct.simps(2) list.set_intros(2))
  qed
qed
then
```

```
  show ?thesis
    using h' ‹h2 ⊢ get_tag_name slot →r ''slot''›
  proof(induct shadow_root_ptrs arbitrary: h2, simp)
    case (Cons shadow_root_ptr shadow_root_ptrs)
    then have "h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →r nodes" and
      "heap_is_wellformed h2" and
      "type_wf h2" and
      "known_ptrs h2"
      by auto
    obtain host h2a h2b h2c host_children shadow_root_children where
      "h2 ⊢ get_host shadow_root_ptr →r host" and
      "h2 ⊢ get_child_nodes (cast host) →r host_children" and
      h2a: "h2 ⊢ forall_M remove host_children →h h2a" and
      "h2a ⊢ get_child_nodes (cast shadow_root_ptr) →r shadow_root_children" and
      h2b: "h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →h h2b" and
      "h2b ⊢ remove_shadow_root host →h h2c" and
      remainder: "h2c ⊢ forall_M(λshadow_root_ptr. Heap_Error_Monad.bind (get_host shadow_root_ptr)
        (λhost. Heap_Error_Monad.bind (get_child_nodes (cast_element_ptr2object_ptr host))
             (λx. Heap_Error_Monad.bind (forall_M remove x)
               (λ_. Heap_Error_Monad.bind (get_child_nodes (cast_shadow_root_ptr2object_ptr shadow_root_ptr))
                    (λx. Heap_Error_Monad.bind (forall_M (append_child (cast_element_ptr2object_ptr host)) x)
                           (λ_. remove_shadow_root host))))))
          shadow_root_ptrs
       →h h'"
      using Cons(3)
      by(auto elim!: bind_returns_heap_E bind_returns_heap_E2[rotated, OF get_host_pure, rotated]
          bind_returns_heap_E2[rotated, OF get_child_nodes_pure, rotated])


    have "h2 ⊢ get_shadow_root host →r Some shadow_root_ptr"
      using ‹h2 ⊢ get_host shadow_root_ptr →r host› shadow_root_host_dual
      using ‹heap_is_wellformed h2› ‹type_wf h2› by blast
    then have "h2a ⊢ get_shadow_root host →r Some shadow_root_ptr"
      using ‹h2 ⊢ forall_M remove host_children →h h2a›
      apply(induct host_children arbitrary: h2)
      by(auto simp add: set_disconnected_nodes_get_shadow_root set_child_nodes_get_shadow_root
          remove_child_locs_def elim!: bind_returns_heap_E
          dest!: reads_writes_separate_forwards[OF get_shadow_root_reads remove_writes])
    then have "h2b ⊢ get_shadow_root host →r Some shadow_root_ptr"
      using ‹h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →h h2b›
      apply(induct shadow_root_children arbitrary: h2a)
      by(auto simp add: set_disconnected_nodes_get_shadow_root set_child_nodes_get_shadow_root
          append_child_def insert_before_locs_def adopt_node_locs_def adopt_node_locs_def remove_child_locs_def
          elim!: bind_returns_heap_E dest!: reads_writes_separate_forwards[OF get_shadow_root_reads insert_before_w
          split: if_splits)

    have "host ≠ slot"
    proof (rule ccontr, simp)
      assume "host = slot"
      show False
        using get_host_valid_tag_name[OF ‹heap_is_wellformed h2› ‹type_wf h2›
            ‹h2 ⊢ get_host shadow_root_ptr →r host›[unfolded ‹host = slot›] ‹h2 ⊢ get_tag_name slot →r
''slot''›]
        by(simp)
    qed

    have "heap_is_wellformed h2a" and "type_wf h2a" and "known_ptrs h2a"
      using ‹heap_is_wellformed h2› and ‹type_wf h2› and ‹known_ptrs h2›
        ‹h2 ⊢ forall_M remove host_children →h h2a›
      apply(induct host_children arbitrary: h2)
      apply(auto simp add: remove_def elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
      apply(auto simp add: remove_def elim!: bind_returns_heap_E
```

```
      bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
  apply(auto simp add: remove_def elim!: bind_returns_heap_E
      bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
  apply(auto simp add: remove_def elim!: bind_returns_heap_E
      bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
  using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
    remove_child_preserves_known_ptrs apply metis
  apply(auto simp add: remove_def elim!: bind_returns_heap_E
      bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
  using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
    remove_child_preserves_known_ptrs apply metis
  apply(auto simp add: remove_def elim!: bind_returns_heap_E
      bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
  using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
    remove_child_preserves_known_ptrs apply metis
  done
then
have "heap_is_wellformed h2b" and "type_wf h2b" and "known_ptrs h2b"
  using ‹h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →_h h2b›
  apply(induct shadow_root_children arbitrary: h2a)
  apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
  apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
  apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
  apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
  using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
    insert_before_preserves_known_ptrs apply metis
  apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
  using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
    insert_before_preserves_known_ptrs apply metis
  apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
  using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
    insert_before_preserves_known_ptrs apply metis
  done
then
have "heap_is_wellformed h2c" and "type_wf h2c" and "known_ptrs h2c"
  using remove_shadow_root_preserves ‹h2b ⊢ remove_shadow_root host →_h h2c›
  by blast+

moreover
have "h2a ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes"
  using ‹h2 ⊢ get_child_nodes (cast_element_ptr2object_ptr slot) →_r nodes›
  using ‹h2 ⊢ forall_M remove host_children →_h h2a›
  using ‹h2 ⊢ get_child_nodes (cast host) →_r host_children›
  using ‹heap_is_wellformed h2› ‹type_wf h2› ‹known_ptrs h2›
proof (induct host_children arbitrary: h2, simp)
  case (Cons a host_children)
  obtain h21 where "h2 ⊢ remove a →_h h21" and
    "h21 ⊢ forall_M remove host_children →_h h2a"
    using Cons(3)
    by(auto elim!: bind_returns_heap_E)

  have "heap_is_wellformed h21" and "type_wf h21" and "known_ptrs h21"
    using ‹heap_is_wellformed h2› and ‹type_wf h2› and ‹known_ptrs h2› ‹h2 ⊢ remove a →_h h21›
    apply(auto simp add: remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
      remove_child_preserves_known_ptrs apply (metis)
    using ‹heap_is_wellformed h2› and ‹type_wf h2› and ‹known_ptrs h2› ‹h2 ⊢ remove a →_h h21›
    apply(auto simp add: remove_def elim!: bind_returns_heap_E
        bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
    using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
      remove_child_preserves_known_ptrs apply (metis)
    using ‹heap_is_wellformed h2› and ‹type_wf h2› and ‹known_ptrs h2› ‹h2 ⊢ remove a →_h h21›
```

```
      apply(auto simp add: remove_def elim!: bind_returns_heap_E
          bind_returns_heap_E2[rotated, OF get_parent_pure, rotated] split: option.splits)[1]
      using remove_child_heap_is_wellformed_preserved remove_child_preserves_type_wf
        remove_child_preserves_known_ptrs apply (metis)
      done
    have "h2 ⊢ get_parent a →_r Some (cast host)"
      using ‹h2 ⊢ get_child_nodes (cast_{element_ptr2object_ptr} host) →_r a # host_children›
      using ‹heap_is_wellformed h2› ‹type_wf h2› ‹known_ptrs h2› child_parent_dual
      using heap_is_wellformed_def by auto
    then have "h21 ⊢ get_child_nodes (cast_{element_ptr2object_ptr} slot) →_r nodes"
      using ‹h2 ⊢ get_child_nodes (cast_{element_ptr2object_ptr} slot) →_r nodes› ‹host ≠ slot›
      using ‹h2 ⊢ remove a →_h h21›
      apply(auto simp add: remove_child_locs_def set_disconnected_nodes_get_child_nodes
          dest!: reads_writes_preserved[OF get_child_nodes_reads remove_writes])[1]
      by (meson cast_{element_ptr2node_ptr}_inject cast_{node_ptr2object_ptr}_inject
          set_child_nodes_get_child_nodes_different_pointers)
    moreover have "h21 ⊢ get_child_nodes (cast_{element_ptr2object_ptr} host) →_r host_children"
      using ‹h2 ⊢ remove a →_h h21›  remove_removes_child[OF ‹heap_is_wellformed h2› ‹type_wf h2›
          ‹known_ptrs h2› ‹h2 ⊢ get_child_nodes (cast_{element_ptr2object_ptr} host) →_r a # host_children›]
      by blast
    ultimately show ?case
      using ‹heap_is_wellformed h21› and ‹type_wf h21› and ‹known_ptrs h21›
        ‹h21 ⊢ forall_M remove host_children →_h h2a› Cons(1)
      using Cons.prems(3) Cons.prems(4) Cons.prems(5) Cons.prems(6) heap_is_wellformed_def
        ‹h2 ⊢ remove a →_h h21› remove_removes_child
      by blast
  qed

  then
  have "h2b ⊢ get_child_nodes (cast_{element_ptr2object_ptr} slot) →_r nodes"
    using ‹h2a ⊢ forall_M (append_child (cast host)) shadow_root_children →_h h2b›
    using ‹h2a ⊢ get_child_nodes (cast shadow_root_ptr) →_r shadow_root_children›
    using ‹heap_is_wellformed h2a› ‹type_wf h2a› ‹known_ptrs h2a›
  proof(induct shadow_root_children arbitrary: h2a, simp)
    case (Cons a shadow_root_children)
    obtain h2a1 where "h2a ⊢ append_child (cast_{element_ptr2object_ptr} host) a →_h h2a1" and
      "h2a1 ⊢ forall_M (append_child (cast_{element_ptr2object_ptr} host)) (shadow_root_children) →_h h2b"
      using Cons(3)
      by(auto elim!: bind_returns_heap_E)

    have "heap_is_wellformed h2a1" and "type_wf h2a1" and "known_ptrs h2a1"
      using ‹heap_is_wellformed h2a› and ‹type_wf h2a› and ‹known_ptrs h2a›
        ‹h2a ⊢ append_child (cast_{element_ptr2object_ptr} host) a →_h h2a1›
      apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
      using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
        insert_before_preserves_known_ptrs apply metis
      using ‹heap_is_wellformed h2a› and ‹type_wf h2a› and ‹known_ptrs h2a›
        ‹h2a ⊢ append_child (cast_{element_ptr2object_ptr} host) a →_h h2a1›
      apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
      using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
        insert_before_preserves_known_ptrs apply metis
      using ‹heap_is_wellformed h2a› and ‹type_wf h2a› and ‹known_ptrs h2a›
        ‹h2a ⊢ append_child (cast_{element_ptr2object_ptr} host) a →_h h2a1›
      apply(auto simp add: append_child_def elim!: bind_returns_heap_E)[1]
      using insert_before_heap_is_wellformed_preserved insert_before_preserves_type_wf
        insert_before_preserves_known_ptrs apply metis
      done
    moreover have "h2a1 ⊢ get_child_nodes (cast shadow_root_ptr) →_r shadow_root_children"
      using ‹h2a ⊢ get_child_nodes (cast shadow_root_ptr) →_r a # shadow_root_children›
      using insert_before_removes_child
        ‹h2a ⊢ append_child (cast_{element_ptr2object_ptr} host) a →_h h2a1›[unfolded append_child_def]
      using ‹heap_is_wellformed h2a› ‹type_wf h2a› ‹known_ptrs h2a›
      using cast_document_ptr_not_node_ptr(2)
```

```
      using cast_shadow_root_ptr_not_node_ptr(2) by blast
    moreover have "h2a ⊢ get_parent a →r Some (cast shadow_root_ptr)"
      using ‹h2a ⊢ get_child_nodes (cast_{shadow_root_ptr2object_ptr} shadow_root_ptr) →r a # shadow_root_children›
      using ‹heap_is_wellformed h2a› ‹type_wf h2a› ‹known_ptrs h2a› child_parent_dual
      using heap_is_wellformed_def by auto
    then have "h2a1 ⊢ get_child_nodes (cast slot) →r nodes"
      using ‹h2a ⊢ get_child_nodes (cast_{element_ptr2object_ptr} slot) →r nodes›
      using ‹h2a ⊢ append_child (cast_{element_ptr2object_ptr} host) a →h h2a1› ‹host ≠ slot›
      apply(auto simp add:  set_disconnected_nodes_get_child_nodes append_child_def
          insert_before_locs_def adopt_node_locs_def adopt_node_locs_def remove_child_locs_def
          elim!: bind_returns_heap_E dest!: reads_writes_preserved[OF get_child_nodes_reads
            insert_before_writes])[1]
      using set_child_nodes_get_child_nodes_different_pointers cast_{element_ptr2node_ptr}_inject
        cast_{node_ptr2object_ptr}_inject cast_document_ptr_not_node_ptr(2)
      by (metis cast_shadow_root_ptr_not_node_ptr(2))+
    ultimately
    show ?case
      using Cons(1) ‹h2a1 ⊢ forall_M (append_child (cast_{element_ptr2object_ptr} host)) shadow_root_children
→h h2b›
      by blast
  qed
  then
  have "h2c ⊢ get_child_nodes (cast_{element_ptr2object_ptr} slot) →r nodes"
    using ‹h2b ⊢ remove_shadow_root host →h h2c›
    apply(auto simp add: remove_shadow_root_get_child_nodes_different_pointers
        dest!: reads_writes_separate_forwards[OF get_child_nodes_reads remove_shadow_root_writes])[1]
    by (meson cast_shadow_root_ptr_not_node_ptr(2)
        local.remove_shadow_root_get_child_nodes_different_pointers)

  moreover
  have "h2a ⊢ get_tag_name slot →r ''slot''"
    using h2a ‹h2 ⊢ get_tag_name slot →r ''slot''›
    apply(induct host_children arbitrary: h2)
    by(auto simp add: remove_child_locs_def set_disconnected_nodes_get_tag_name
        set_child_nodes_get_tag_name dest!: reads_writes_separate_forwards[OF get_tag_name_reads remove_writes]
        elim!: bind_returns_heap_E)
  then
  have "h2b ⊢ get_tag_name slot →r ''slot''"
    using h2b
    apply(induct shadow_root_children arbitrary: h2a)
    by(auto simp add: append_child_def insert_before_locs_def adopt_node_locs_def adopt_node_locs_def
        remove_child_locs_def set_disconnected_nodes_get_tag_name set_child_nodes_get_tag_name
        dest!: reads_writes_separate_forwards[OF get_tag_name_reads insert_before_writes]
        elim!: bind_returns_heap_E split: if_splits)
  then
  have "h2c ⊢ get_tag_name slot →r ''slot''"
    using ‹h2b ⊢ remove_shadow_root host →h h2c›
    by(auto simp add: remove_shadow_root_get_tag_name
        dest!: reads_writes_separate_forwards[OF get_tag_name_reads remove_shadow_root_writes])

  ultimately show ?case
    using Cons(1) remainder
    by auto
 qed
qed
end
interpretation i_assigned_nodes_component?: l_assigned_nodes_component_{Shadow_DOM}
  type_wf get_tag_name get_tag_name_locs known_ptr get_child_nodes get_child_nodes_locs
  get_disconnected_nodes get_disconnected_nodes_locs get_shadow_root get_shadow_root_locs
  heap_is_wellformed parent_child_rel heap_is_wellformed_{Core_DOM} get_host get_host_locs
  get_disconnected_document get_disconnected_document_locs get_parent get_parent_locs
  get_mode get_mode_locs get_attribute get_attribute_locs first_in_tree_order find_slot
  assigned_slot known_ptrs to_tree_order assigned_nodes assigned_nodes_flatten flatten_dom
```

```
  get_root_node get_root_node_locs remove insert_before insert_before_locs append_child
  remove_shadow_root remove_shadow_root_locs set_shadow_root set_shadow_root_locs remove_child
  remove_child_locs get_component is_strongly_dom_component_safe is_weakly_dom_component_safe
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name get_owner_document set_disconnected_nodes set_disconnected_nodes_locs
  adopt_node adopt_node_locs set_child_nodes set_child_nodes_locs
  by(auto simp add: l_assigned_nodes_component_Shadow_DOM_def instances)
declare l_assigned_nodes_component_Shadow_DOM_axioms [instances]
```

get_owner_document

```
locale l_get_owner_document_component_Shadow_DOM =
  l_get_owner_document_wf_Shadow_DOM +
  l_get_component_Core_DOM
begin
lemma get_owner_document_is_component_unsafe:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_owner_document ptr →_r owner_document"
  assumes "¬is_document_ptr_kind |h ⊢ get_root_node ptr|_r"
  shows "set |h ⊢ get_component ptr|_r ∩ set |h ⊢ get_component (cast owner_document)|_r = {}"
proof -
  have "owner_document |∈| document_ptr_kinds h"
    using assms(1) assms(2) assms(3) assms(4) get_owner_document_owner_document_in_heap by blast
  have "ptr |∈| object_ptr_kinds h"
    by (meson assms(4) is_OK_returns_result_I local.get_owner_document_ptr_in_heap)
  obtain root where root: "h ⊢ get_root_node ptr →_r root"
    by (meson assms(1) assms(2) assms(3) assms(4) is_OK_returns_result_I
        local.get_owner_document_ptr_in_heap local.get_root_node_ok returns_result_select_result)
  then obtain to where to: "h ⊢ to_tree_order root →_r to"
    by (meson assms(1) assms(2) assms(3) is_OK_returns_result_E local.get_root_node_root_in_heap
        local.to_tree_order_ok)
  then have "∀p ∈ set to. ¬is_document_ptr_kind p"
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(5) document_ptr_casts_commute3
        local.to_tree_order_node_ptrs node_ptr_no_document_ptr_cast root select_result_I2)
  then have "cast owner_document ∉ set |h ⊢ get_component ptr|_r"
    by (metis (no_types, lifting) assms(1) assms(2) assms(3) assms(5) document_ptr_document_ptr_cast
        is_OK_returns_result_I l_get_component_Core_DOM.get_component_ok local.get_component_root_node_same
        local.get_root_node_not_node_same local.get_root_node_ptr_in_heap local.l_get_component_Core_DOM_axioms
        node_ptr_no_document_ptr_cast returns_result_select_result root select_result_I2)
  then have "|h ⊢ get_component ptr|_r ≠ |h ⊢ get_component (cast owner_document)|_r"
    by (metis (no_types, lifting) ‹owner_document |∈| document_ptr_kinds h› assms(1) assms(2) assms(3)
        document_ptr_kinds_commutes l_get_component_Core_DOM.get_component_ok local.get_dom_component_ptr
        local.l_get_component_Core_DOM_axioms returns_result_select_result)
  then show ?thesis
    by (meson ‹owner_document |∈| document_ptr_kinds h› ‹ptr |∈| object_ptr_kinds h› assms(1)
        assms(2) assms(3) document_ptr_kinds_commutes l_get_component_Core_DOM.get_dom_component_no_overlap
        l_get_component_Core_DOM.get_component_ok local.l_get_component_Core_DOM_axioms returns_result_select_resu
qed

end
interpretation i_get_owner_document_component?: l_get_owner_document_component_Shadow_DOM
  type_wf get_disconnected_nodes get_disconnected_nodes_locs known_ptr get_child_nodes
  get_child_nodes_locs DocumentClass.known_ptr get_parent get_parent_locs get_root_node_si
  get_root_node_si_locs CD.a_get_owner_document get_host get_host_locs get_owner_document
  get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs heap_is_wellformed
  parent_child_rel heap_is_wellformed_Core_DOM get_disconnected_document get_disconnected_document_locs
  known_ptrs get_ancestors_si get_ancestors_si_locs to_tree_order get_component
  is_strongly_dom_component_safe is_weakly_dom_component_safe get_root_node get_root_node_locs
  get_ancestors get_ancestors_locs get_element_by_id get_elements_by_class_name
  get_elements_by_tag_name
  by(auto simp add: l_get_owner_document_component_Shadow_DOM_def instances)
declare l_get_owner_document_component_Shadow_DOM_axioms [instances]
```

```
definition is_shadow_root_component :: "(_) object_ptr list ⇒ bool"
  where
    "is_shadow_root_component c = is_shadow_root_ptr_kind (hd c)"

end
```

3 Example

3.1 Testing fancy_tabs (fancy_tabs)

This theory contains the test cases for fancy_tabs.

theory *fancy_tabs*
imports
"Shadow_DOM.Shadow_DOM"
begin

definition *fancy_tabs_heap :: "heap$_{final}$"* **where**
"fancy_tabs_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html (Some (cast (element_ptr.Ref 1))) [])),
(cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref 4)] fmempty None)),
(cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3)] fmempty None)),
(cast (element_ptr.Ref 3), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty None)),
(cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Global%20News'')),
(cast (element_ptr.Ref 4), cast (create_element_obj ''body'' [cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7), cast (element_ptr.Ref 8), cast (element_ptr.Ref 9), cast (element_ptr.Ref 10), cast (element_ptr.Ref 30)] fmempty None)),
(cast (element_ptr.Ref 5), cast (create_element_obj ''h1'' [cast (character_data_ptr.Ref 2)] fmempty None)),
(cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''Global%20News'')),
(cast (element_ptr.Ref 6), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 3)] fmempty None)),
(cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''Search'')),
(cast (element_ptr.Ref 7), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 4)] fmempty None)),
(cast (character_data_ptr.Ref 4), cast (create_character_data_obj ''Sign%20In'')),
(cast (element_ptr.Ref 8), cast (create_element_obj ''br'' [] fmempty None)),
(cast (element_ptr.Ref 9), cast (create_element_obj ''br'' [] fmempty None)),
(cast (element_ptr.Ref 10), cast (create_element_obj ''fancy-tabs'' [cast (element_ptr.Ref 11), cast (element_ptr.Ref 12), cast (element_ptr.Ref 13), cast (element_ptr.Ref 14), cast (element_ptr.Ref 15), cast (element_ptr.Ref 22)] (fmap_of_list [(''background'', '''')]) (Some (cast (shadow_root_ptr.Ref 1))))),
(cast (element_ptr.Ref 11), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 5)] (fmap_of_list [(''slot'', ''title'')]) None)),
(cast (character_data_ptr.Ref 5), cast (create_character_data_obj ''Politics'')),
(cast (element_ptr.Ref 12), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 6)] (fmap_of_list [(''slot'', ''title''), (''selected'', '''')]) None)),
(cast (character_data_ptr.Ref 6), cast (create_character_data_obj ''Sports'')),
(cast (element_ptr.Ref 13), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 7)] (fmap_of_list [(''slot'', ''title'')]) None)),
(cast (character_data_ptr.Ref 7), cast (create_character_data_obj ''Culture'')),
(cast (element_ptr.Ref 14), cast (create_element_obj ''section'' [cast (character_data_ptr.Ref 8)] fmempty None)),
(cast (character_data_ptr.Ref 8), cast (create_character_data_obj ''content%20panel%201'')),
(cast (element_ptr.Ref 15), cast (create_element_obj ''ul'' [cast (element_ptr.Ref 16), cast (element_ptr.Ref 18), cast (element_ptr.Ref 20)] fmempty None)),
(cast (element_ptr.Ref 16), cast (create_element_obj ''li'' [cast (character_data_ptr.Ref 9), cast (element_ptr 17)] fmempty None)),
(cast (character_data_ptr.Ref 9), cast (create_character_data_obj ''News%20Item%201'')),
(cast (element_ptr.Ref 17), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 10)] fmempty None)),
(cast (character_data_ptr.Ref 10), cast (create_character_data_obj ''Share'')),

```
    (cast (element_ptr.Ref 18), cast (create_element_obj ''li'' [cast (character_data_ptr.Ref 11), cast
(element_ptr.Ref 19)] fmempty None)),
    (cast (character_data_ptr.Ref 11), cast (create_character_data_obj ''News%20Item%202'')),
    (cast (element_ptr.Ref 19), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 12)] fmempty
None)),
    (cast (character_data_ptr.Ref 12), cast (create_character_data_obj ''Share'')),
    (cast (element_ptr.Ref 20), cast (create_element_obj ''li'' [cast (character_data_ptr.Ref 13), cast
(element_ptr.Ref 21)] fmempty None)),
    (cast (character_data_ptr.Ref 13), cast (create_character_data_obj ''News%20Item%203'')),
    (cast (element_ptr.Ref 21), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 14)] fmempty
None)),
    (cast (character_data_ptr.Ref 14), cast (create_character_data_obj ''Share'')),
    (cast (element_ptr.Ref 22), cast (create_element_obj ''section'' [cast (character_data_ptr.Ref 15)]
fmempty None)),
    (cast (character_data_ptr.Ref 15), cast (create_character_data_obj ''content%20panel%203'')),
    (cast (shadow_root_ptr.Ref 1), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 23), cast (element_ptr.
24), cast (element_ptr.Ref 26), cast (element_ptr.Ref 28), cast (element_ptr.Ref 29)])),
    (cast (element_ptr.Ref 23), cast (create_element_obj ''style'' [cast (character_data_ptr.Ref 16)] fmempty
None)),
    (cast (character_data_ptr.Ref 16), cast (create_character_data_obj ''---shortened---'')),
    (cast (element_ptr.Ref 24), cast (create_element_obj ''div'' [cast (element_ptr.Ref 25)] (fmap_of_list
[(''id'', ''tabs'')]) None)),
    (cast (element_ptr.Ref 25), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''tabsSlot''),
(''name'', ''title'')]) None)),
    (cast (element_ptr.Ref 26), cast (create_element_obj ''div'' [cast (element_ptr.Ref 27)] (fmap_of_list
[(''id'', ''panels'')]) None)),
    (cast (element_ptr.Ref 27), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''panelsSlot'')])
None)),
    (cast (element_ptr.Ref 28), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 17)] fmempty
None)),
    (cast (character_data_ptr.Ref 17), cast (create_character_data_obj ''Previous%20Tab'')),
    (cast (element_ptr.Ref 29), cast (create_element_obj ''button'' [cast (character_data_ptr.Ref 18)] (fmap_of_lis
[(''style'', ''float:right'')]) None)),
    (cast (character_data_ptr.Ref 18), cast (create_character_data_obj ''Next%20Tab'')),
    (cast (element_ptr.Ref 30), cast (create_element_obj ''script'' [] fmempty None))]"
```

definition *fancy_tabs_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option"* **where** *"fancy_tabs_docum*
= Some (cast (document_ptr.Ref 1))"

end

Bibliography

[1] E. Bidelman. Shadow dom v1: Self-contained web components, May 2017. URL https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom.

[2] A. D. Brucker and M. Herzberg. The core DOM. *Archive of Formal Proofs*, dec 2018. ISSN 2150-914x. URL https://www.brucker.ch/bibliography/abstract/brucker.ea-afp-core-dom-2018-a. http://www.isa-afp.org/entries/Core_DOM.html, Formal proof development.

[3] A. D. Brucker and M. Herzberg. A formal semantics of the core DOM in Isabelle/HOL. In *Proceedings of the Web Programming, Design, Analysis, And Implementation (WPDAI) track at WWW 2018*, 2018. URL https://www.brucker.ch/bibliography/abstract/brucker.ea-fdom-2018.

[4] A. D. Brucker and M. Herzberg. Shadow dom: A formal model of the document object model with shadow roots. *Archive of Formal Proofs*, Sept. 2020. ISSN 2150-914x. URL http://www.brucker.ch/bibliography/abstract/brucker.ea-afp-shadow-dom-2020. http://www.isa-afp.org/entries/Shadow_DOM.html, Formal proof development.

[5] A. D. Brucker and M. Herzberg. A formally verified model of web components. In S.-S. Jongmans and F. Arbab, editors, *Formal Aspects of Component Software (FACS)*, number 12018 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2020. ISBN 3-540-25109-X. doi: 10.1007/978-3-030-40914-2_3. URL http://www.brucker.ch/bibliography/abstract/brucker.ea-web-components-2019.

[6] M. Herzberg. *Formal Foundations for Provably Safe Web Components*. PhD thesis, The University of Sheffield, 2020.

[7] WHATWG. DOM – living standard, Feb. 2019. URL https://dom.spec.whatwg.org/commit-snapshots/7fa83673430f767d329406d0aed901f296332216/. Last Updated 11 February 2019.