Shadow DOM

A Formal Model of the Document Object Model *with Shadow Roots*

Achim D. Brucker* Michael Herzberg†

February 6, 2026

*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

In this AFP entry, we extend our formalization of the core DOM (AFP entry Core_DOM) with *Shadow Roots*. Shadow roots are a recent proposal of the web community to support a component-based development approach for client-side web applications.

Shadow roots are a significant extension to the DOM standard and, as web standards are condemned to be backward compatible, such extensions often result in complex specification that may contain unwanted subtleties that can be detected by a formalization.

Our Isabelle/HOL formalization is, in the sense of object-orientation, an extension of our formalization of the core DOM and enjoys the same basic properties, i.e., it is 1. *extensible*, i.e., can be extended without the need of re-proving already proven properties and 2. *executable*, i.e., we can generate executable code from our specification. We exploit the executability to show that our formalization complies to the official standard of the W3C, respectively, the WHATWG.

Keywords: Document Object Model, DOM, Shadow Root, Web Component, Formal Semantics, Isabelle/HOL

Contents

1 Introduction **7**

2 The Shadow DOM **9**
2.1 The Shadow DOM Data Model (ShadowRootClass) 9
2.2 Shadow Root Monad (ShadowRootMonad) 15
2.3 The Shadow DOM (Shadow_DOM) 22

3 Test Suite **97**
3.1 Shadow DOM Base Tests (Shadow_DOM_BaseTest) 97
3.2 Testing slots (slots) 102
3.3 Testing slots_fallback (slots_fallback) 118
3.4 Testing Document_adoptNode (Shadow_DOM_Document_adoptNode) 126
3.5 Testing Document_getElementById (Shadow_DOM_Document_getElementById) 127
3.6 Testing Node_insertBefore (Shadow_DOM_Node_insertBefore) 131
3.7 Testing Node_removeChild (Shadow_DOM_Node_removeChild) 133
3.8 Shadow DOM Tests (Shadow_DOM_Tests) 135

1 Introduction

In a world in which more and more applications are offered as services on the internet, web browsers start to take on a similarly central role in our daily IT infrastructure as operating systems. Thus, web browsers should be developed as rigidly and formally as operating systems. While formal methods are a well-established technique in the development of operating systems (see, e. g., Klein [13] for an overview of formal verification of operating systems), there are few proposals for improving the development of web browsers using formal approaches [1, 10, 12, 14].

In [3], we formalized the core of the Document Object Model (DOM) in Isabelle/HOL. The DOM [15, 16] is *the* central data structure of all modern web browsers. In this work, we extend the formalization presented in [3] with support for *shadow trees*. Shadow trees are a recent addition to the DOM standard [16] that promise support for web components. As we will see, this promise is not fully achieved and, for example, the DOM standard itself does not formally define what a component should be. In this work, we focus on a standard compliant representation of the DOM with shadow trees. As [3], our formalization has the following properties:

- It provides a *consistency guarantee.* Since all definitions in our formal semantics are conservative and all rules are derived, the logical consistency of the DOM node-tree is reduced to the consistency of HOL.
- It serves as a *technical basis for a proof system.* Based on the derived rules and specific setup of proof tactics over node-trees, our formalization provides a generic proof environment for the verification of programs manipulating node-trees.
- It is *executable*, which allows to validate its compliance to the standard by evaluating the compliance test suite on the formal model and
- It is *extensible* in the sense of [2, 8], i. e., properties proven over the core DOM do not need to be re-proven for object-oriented extensions such as the HTML document model.

In this AFP entry, we limit ourselves to the faithful formalization of the DOM. As the DOM standard does not formally define web components, we address the question of formally defining web components and discussing their safety properties elsewhere [5, 6].

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 and additional explanations to [6, 11]. The structure follows the theory dependencies (see Figure 1.1): first, we formalize the DOM with Shadow Roots (chapter 2) and then formalize we the relevant compliance test cases in chapter 3.

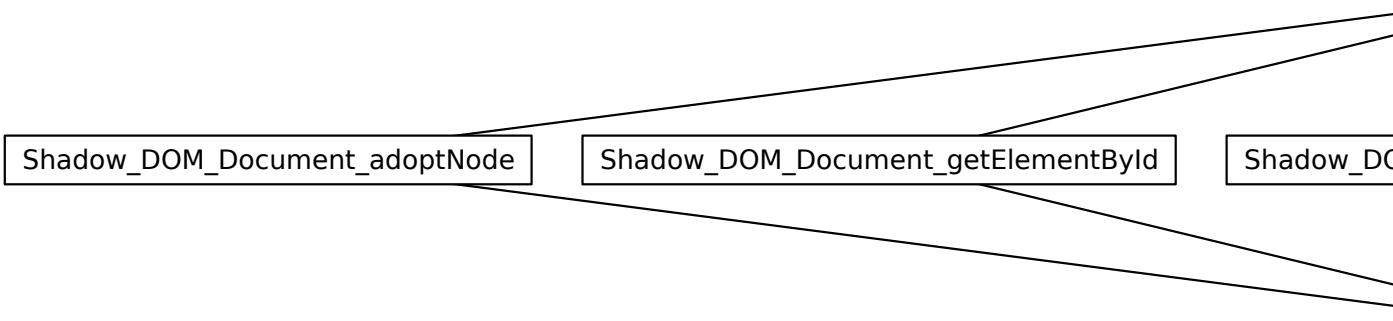

Figure 1.1: The Dependency Graph of the Isabelle Theories.

2 The Shadow DOM

In this chapter, we introduce the formalization of the core DOM *with Shadow Roots*, i.e., the most important algorithms for querying or modifying the Shadow DOM, as defined in the standard.

2.1 The Shadow DOM Data Model (ShadowRootClass)

theory *ShadowRootClass*
  imports
    Core_DOM.ShadowRootPointer
    Core_DOM.DocumentClass
begin

2.1.1 ShadowRoot

datatype *shadow_root_mode = Open | Closed*
record *('node_ptr, 'element_ptr, 'character_data_ptr) RShadowRoot = RObject +*
  nothing :: unit
  mode :: shadow_root_mode
  child_nodes :: "('node_ptr, 'element_ptr, 'character_data_ptr) node_ptr list"
type_synonym *('node_ptr, 'element_ptr, 'character_data_ptr, 'ShadowRoot) ShadowRoot*
  = "('node_ptr, 'element_ptr, 'character_data_ptr, 'ShadowRoot option) RShadowRoot_scheme"
register_default_tvars *"('node_ptr, 'element_ptr, 'character_data_ptr, 'ShadowRoot) ShadowRoot"*
type_synonym *('node_ptr, 'element_ptr, 'character_data_ptr, 'shadow_root_ptr, 'Object, 'Node,*
    'Element, 'CharacterData, 'Document,
    'ShadowRoot) Object
  = "('node_ptr, 'element_ptr, 'character_data_ptr, 'shadow_root_ptr, ('node_ptr, 'element_ptr,
'character_data_ptr, 'ShadowRoot option)
      RShadowRoot_ext + 'Object, 'Node, 'Element, 'CharacterData, 'Document) Object"
register_default_tvars *"('node_ptr, 'element_ptr, 'character_data_ptr, 'shadow_root_ptr, 'Object,*
'Node, 'Element, 'CharacterData,
    'Document, 'ShadowRoot) Object"

type_synonym *('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr,*
    'shadow_root_ptr, 'Object, 'Node,
    'Element, 'CharacterData, 'Document, 'ShadowRoot) heap
  = "('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr, 'shadow_root_ptr,
('node_ptr, 'element_ptr,
      'character_data_ptr, 'ShadowRoot option) RShadowRoot_ext + 'Object, 'Node, 'Element,
'CharacterData, 'Document) heap"
register_default_tvars *"('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr,*
'shadow_root_ptr, 'Object,
    'Node, 'Element, 'CharacterData, 'Document, 'ShadowRoot) heap"
type_synonym *"heap$_{final}$" = "(unit, unit, unit, unit, unit, unit, unit, unit, unit, unit, unit) heap"*

definition *shadow_root_ptr_kinds :: "(_) heap ⇒ (_) shadow_root_ptr fset"*
  where
    "shadow_root_ptr_kinds heap =
the |‘| (cast |‘| (ffilter is_shadow_root_ptr_kind (object_ptr_kinds heap)))"

lemma *shadow_root_ptr_kinds_simp [simp]:*
  "shadow_root_ptr_kinds (Heap (fmupd (cast shadow_root_ptr) shadow_root (the_heap h))) =
{|shadow_root_ptr|} |∪| shadow_root_ptr_kinds h"
  ⟨*proof*⟩

```
definition shadow_root_ptrs :: "(_) heap ⇒ (_) shadow_root_ptr fset"
  where
    "shadow_root_ptrs heap = ffilter is_shadow_root_ptr (shadow_root_ptr_kinds heap)"

definition cast_Object2ShadowRoot :: "(_) Object ⇒ (_) ShadowRoot option"
  where
    "cast_Object2ShadowRoot obj = (case RObject.more obj of
      Inr (Inr (Inl shadow_root)) ⇒ Some (RObject.extend (RObject.truncate obj) shadow_root)
    | _ ⇒ None)"
adhoc_overloading cast ⇌ cast_Object2ShadowRoot

definition cast_ShadowRoot2Object:: "(_) ShadowRoot ⇒ (_) Object"
  where
    "cast_ShadowRoot2Object shadow_root =
(RObject.extend (RObject.truncate shadow_root) (Inr (Inr (Inl (RObject.more shadow_root)))))"
adhoc_overloading cast ⇌ cast_ShadowRoot2Object

definition is_shadow_root_kind :: "(_) Object ⇒ bool"
  where
    "is_shadow_root_kind ptr ⟷ cast_Object2ShadowRoot ptr ≠ None"

lemma shadow_root_ptr_kinds_heap_upd [simp]:
  "shadow_root_ptr_kinds (Heap (fmupd (cast shadow_root_ptr) shadow_root (the_heap h))) =
{|shadow_root_ptr|} |∪| shadow_root_ptr_kinds h"
  ⟨proof⟩

lemma shadow_root_ptr_kinds_commutes [simp]:
  "cast shadow_root_ptr |∈| object_ptr_kinds h ⟷ shadow_root_ptr |∈| shadow_root_ptr_kinds h"
⟨proof⟩

definition get_ShadowRoot :: "(_) shadow_root_ptr ⇒ (_) heap ⇒ (_) ShadowRoot option"
  where
    "get_ShadowRoot shadow_root_ptr h = Option.bind (get (cast shadow_root_ptr) h) cast"
adhoc_overloading get ⇌ get_ShadowRoot

locale l_type_wf_def_ShadowRoot
begin
definition a_type_wf :: "(_) heap ⇒ bool"
  where
    "a_type_wf h = (DocumentClass.type_wf h ∧ (∀shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h).
get_ShadowRoot shadow_root_ptr h ≠ None))"
end
global_interpretation l_type_wf_def_ShadowRoot defines type_wf = a_type_wf ⟨proof⟩
lemmas type_wf_defs = a_type_wf_def

locale l_type_wf_ShadowRoot = l_type_wf type_wf for type_wf :: "((_) heap ⇒ bool)" +
  assumes type_wf_ShadowRoot: "type_wf h ⟹ ShadowRootClass.type_wf h"

sublocale l_type_wf_ShadowRoot ⊆ l_type_wf_Document
  ⟨proof⟩

lemma type_wf_implies_previous: "type_wf h ⟹ DocumentClass.type_wf h"
  ⟨proof⟩

locale l_get_ShadowRoot_lemmas = l_type_wf_ShadowRoot
begin
sublocale l_get_Document_lemmas ⟨proof⟩
lemma get_ShadowRoot_type_wf:
  assumes "type_wf h"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟷ get_ShadowRoot shadow_root_ptr h ≠ None"
  ⟨proof⟩
end
```

global_interpretation *l_get$_{ShadowRoot}$_lemmas type_wf* ⟨*proof*⟩

definition *put$_{ShadowRoot}$ :: "(_) shadow_root_ptr ⇒ (_) ShadowRoot ⇒ (_) heap ⇒ (_) heap"*
where
"put$_{ShadowRoot}$ shadow_root_ptr shadow_root = put (cast shadow_root_ptr) (cast shadow_root)"
adhoc_overloading *put ⇌ put$_{ShadowRoot}$*

lemma *put$_{ShadowRoot}$_ptr_in_heap:*
assumes *"put$_{ShadowRoot}$ shadow_root_ptr shadow_root h = h'"*
shows *"shadow_root_ptr |∈| shadow_root_ptr_kinds h'"*
⟨*proof*⟩

lemma *put$_{ShadowRoot}$_put_ptrs:*
assumes *"put$_{ShadowRoot}$ shadow_root_ptr shadow_root h = h'"*
shows *"object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast shadow_root_ptr|}"*
⟨*proof*⟩

lemma *cast$_{ShadowRoot2Object}$_inject [simp]: "cast$_{ShadowRoot2Object}$ x = cast$_{ShadowRoot2Object}$ y ⟷ x = y"*
⟨*proof*⟩

lemma *cast$_{Object2ShadowRoot}$_none [simp]:*
"cast$_{Object2ShadowRoot}$ obj = None ⟷ ¬ (∃shadow_root. cast$_{ShadowRoot2Object}$ shadow_root = obj)"
⟨*proof*⟩

lemma *cast$_{Object2ShadowRoot}$_some [simp]:*
"cast$_{Object2ShadowRoot}$ obj = Some shadow_root ⟷ cast shadow_root = obj"
⟨*proof*⟩

lemma *cast$_{Object2ShadowRoot}$_inv [simp]: "cast$_{Object2ShadowRoot}$ (cast$_{ShadowRoot2Object}$ shadow_root) = Some shadow_root"*
⟨*proof*⟩

lemma *cast_shadow_root_not_node [simp]:*
"cast$_{ShadowRoot2Object}$ shadow_root ≠ cast$_{Node2Object}$ node"
"cast$_{Node2Object}$ node ≠ cast$_{ShadowRoot2Object}$ shadow_root"
⟨*proof*⟩

lemma *get_shadow_root_ptr_simp1 [simp]:*
"get$_{ShadowRoot}$ shadow_root_ptr (put$_{ShadowRoot}$ shadow_root_ptr shadow_root h) = Some shadow_root"
⟨*proof*⟩
lemma *get_shadow_root_ptr_simp2 [simp]:*
"shadow_root_ptr ≠ shadow_root_ptr'
⟹ get$_{ShadowRoot}$ shadow_root_ptr (put$_{ShadowRoot}$ shadow_root_ptr' shadow_root h) = get$_{ShadowRoot}$ shadow_root_ptr h"
⟨*proof*⟩

lemma *get_shadow_root_ptr_simp3 [simp]:*
"get$_{Element}$ element_ptr (put$_{ShadowRoot}$ shadow_root_ptr f h) = get$_{Element}$ element_ptr h"
⟨*proof*⟩
lemma *get_shadow_root_ptr_simp4 [simp]:*
"get$_{ShadowRoot}$ shadow_root_ptr (put$_{Element}$ element_ptr f h) = get$_{ShadowRoot}$ shadow_root_ptr h"
⟨*proof*⟩
lemma *get_shadow_root_ptr_simp5 [simp]:*
"get$_{CharacterData}$ character_data_ptr (put$_{ShadowRoot}$ shadow_root_ptr f h) = get$_{CharacterData}$ character_data_ptr h"
⟨*proof*⟩
lemma *get_shadow_root_ptr_simp6 [simp]:*
"get$_{ShadowRoot}$ shadow_root_ptr (put$_{CharacterData}$ character_data_ptr f h) = get$_{ShadowRoot}$ shadow_root_ptr h"
⟨*proof*⟩

lemma *get_shadow_root_ptr_simp7 [simp]:*
"get$_{Document}$ document_ptr (put$_{ShadowRoot}$ shadow_root_ptr f h) = get$_{Document}$ document_ptr h"
⟨*proof*⟩
lemma *get_shadow_root_ptr_simp8 [simp]:*
"get$_{ShadowRoot}$ shadow_root_ptr (put$_{Document}$ document_ptr f h) = get$_{ShadowRoot}$ shadow_root_ptr h"
⟨*proof*⟩

lemma *new$_{Element}$_get$_{ShadowRoot}$ [simp]:*
assumes *"new$_{Element}$ h = (new_element_ptr, h')"*
shows *"get$_{ShadowRoot}$ ptr h = get$_{ShadowRoot}$ ptr h'"*
⟨*proof*⟩

lemma *new$_{CharacterData}$_get$_{ShadowRoot}$ [simp]:*
assumes *"new$_{CharacterData}$ h = (new_character_data_ptr, h')"*
shows *"get$_{ShadowRoot}$ ptr h = get$_{ShadowRoot}$ ptr h'"*
⟨*proof*⟩

lemma *new$_{Document}$_get$_{ShadowRoot}$ [simp]:*
assumes *"new$_{Document}$ h = (new_document_ptr, h')"*
shows *"get$_{ShadowRoot}$ ptr h = get$_{ShadowRoot}$ ptr h'"*
⟨*proof*⟩

abbreviation *"create_shadow_root_obj mode_arg child_nodes_arg*
≡ (| RObject.nothing = (), RShadowRoot.nothing = (), mode = mode_arg,
RShadowRoot.child_nodes = child_nodes_arg, ... = None |)"

definition *new$_{ShadowRoot}$ :: "(_)heap ⇒ ((_) shadow_root_ptr × (_) heap)"*
where
"new$_{ShadowRoot}$ h = (let new_shadow_root_ptr =
shadow_root_ptr.Ref (Suc (fMax (shadow_root_ptr.the_ref |`| (shadow_root_ptrs h)))) in
(new_shadow_root_ptr, put new_shadow_root_ptr (create_shadow_root_obj Open []) h))"

lemma *new$_{ShadowRoot}$_ptr_in_heap:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"new_shadow_root_ptr |∈| shadow_root_ptr_kinds h'"*
⟨*proof*⟩

lemma *new_shadow_root_ptr_new:*
"shadow_root_ptr.Ref (Suc (fMax (finsert 0 (shadow_root_ptr.the_ref |`| shadow_root_ptrs h)))) |∉|
shadow_root_ptrs h"
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_ptr_not_in_heap:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"new_shadow_root_ptr |∉| shadow_root_ptr_kinds h"*
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_new_ptr:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"*
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_is_shadow_root_ptr:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"is_shadow_root_ptr new_shadow_root_ptr"*
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_get$_{Object}$ [simp]:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
assumes *"ptr ≠ cast new_shadow_root_ptr"*
shows *"get$_{Object}$ ptr h = get$_{Object}$ ptr h'"*

⟨*proof*⟩

lemma *new$_{ShadowRoot}$_get$_{Node}$ [simp]:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"get$_{Node}$ ptr h = get$_{Node}$ ptr h'"*
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_get$_{Element}$ [simp]:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"get$_{Element}$ ptr h = get$_{Element}$ ptr h'"*
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_get$_{CharacterData}$ [simp]:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"get$_{CharacterData}$ ptr h = get$_{CharacterData}$ ptr h'"*
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_get$_{Document}$ [simp]:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
shows *"get$_{Document}$ ptr h = get$_{Document}$ ptr h'"*
⟨*proof*⟩

lemma *new$_{ShadowRoot}$_get$_{ShadowRoot}$ [simp]:*
assumes *"new$_{ShadowRoot}$ h = (new_shadow_root_ptr, h')"*
assumes *"ptr ≠ new_shadow_root_ptr"*
shows *"get$_{ShadowRoot}$ ptr h = get$_{ShadowRoot}$ ptr h'"*
⟨*proof*⟩

locale *l_known_ptr$_{ShadowRoot}$*
begin
definition *a_known_ptr :: "(_) object_ptr ⇒ bool"*
where
"a_known_ptr ptr = (known_ptr ptr ∨ is_shadow_root_ptr ptr)"

lemma *known_ptr_not_shadow_root_ptr: "a_known_ptr ptr ⟹ ¬is_shadow_root_ptr ptr ⟹ known_ptr ptr"*
⟨*proof*⟩
lemma *known_ptr_new_shadow_root_ptr: "a_known_ptr ptr ⟹ ¬known_ptr ptr ⟹ is_shadow_root_ptr ptr"*
⟨*proof*⟩

end
global_interpretation *l_known_ptr$_{ShadowRoot}$* **defines** *known_ptr = a_known_ptr* ⟨*proof*⟩
lemmas *known_ptr_defs = a_known_ptr_def*

locale *l_known_ptrs$_{ShadowRoot}$ = l_known_ptr known_ptr* **for** *known_ptr :: "(_) object_ptr ⇒ bool"*
begin
definition *a_known_ptrs :: "(_) heap ⇒ bool"*
where
"a_known_ptrs h = (∀ptr ∈ fset (object_ptr_kinds h). known_ptr ptr)"

lemma *known_ptrs_known_ptr: "a_known_ptrs h ⟹ ptr |∈| object_ptr_kinds h ⟹ known_ptr ptr"*
⟨*proof*⟩

lemma *known_ptrs_preserved:*
"object_ptr_kinds h = object_ptr_kinds h' ⟹ a_known_ptrs h = a_known_ptrs h'"
⟨*proof*⟩
lemma *known_ptrs_subset:*
"object_ptr_kinds h' |⊆| object_ptr_kinds h ⟹ a_known_ptrs h ⟹ a_known_ptrs h'"
⟨*proof*⟩
lemma *known_ptrs_new_ptr:*
"object_ptr_kinds h' = object_ptr_kinds h |∪| {|new_ptr|} ⟹ known_ptr new_ptr ⟹
a_known_ptrs h ⟹ a_known_ptrs h'"
⟨*proof*⟩

end
global_interpretation *l_known_ptrs*$_{ShadowRoot}$ *known_ptr* **defines** *known_ptrs = a_known_ptrs* ⟨*proof*⟩
lemmas *known_ptrs_defs = a_known_ptrs_def*

lemma *known_ptrs_is_l_known_ptrs [instances]: "l_known_ptrs known_ptr known_ptrs"*
⟨*proof*⟩

lemma *shadow_root_get_put_1 [simp]:*
"get$_{ShadowRoot}$ *shadow_root_ptr (put*$_{ShadowRoot}$ *shadow_root_ptr shadow_root h) = Some shadow_root"*
⟨*proof*⟩
lemma *shadow_root_different_get_put [simp]:*
"shadow_root_ptr ≠ shadow_root_ptr' ⟹
get$_{ShadowRoot}$ *shadow_root_ptr (put*$_{ShadowRoot}$ *shadow_root_ptr' shadow_root h) = get*$_{ShadowRoot}$ *shadow_root_ptr h"*
⟨*proof*⟩

lemma *shadow_root_get_put_2 [simp]:*
"get$_{Element}$ *element_ptr (put*$_{ShadowRoot}$ *shadow_root_ptr f h) = get*$_{Element}$ *element_ptr h"*
⟨*proof*⟩
lemma *shadow_root_get_put_3 [simp]:*
"get$_{ShadowRoot}$ *element_ptr (put*$_{Element}$ *shadow_root_ptr f h) = get*$_{ShadowRoot}$ *element_ptr h"*
⟨*proof*⟩
lemma *shadow_root_get_put_4 [simp]:*
"get$_{CharacterData}$ *character_data_ptr (put*$_{ShadowRoot}$ *shadow_root_ptr f h) = get*$_{CharacterData}$ *character_data_ptr h"*
⟨*proof*⟩
lemma *shadow_root_get_put_5 [simp]:*
"get$_{ShadowRoot}$ *character_data_ptr (put*$_{CharacterData}$ *shadow_root_ptr f h) = get*$_{ShadowRoot}$ *character_data_ptr h"*
⟨*proof*⟩
lemma *shadow_root_get_put_6 [simp]:*
"get$_{Document}$ *document_ptr (put*$_{ShadowRoot}$ *shadow_root_ptr f h) = get*$_{Document}$ *document_ptr h"*
⟨*proof*⟩
lemma *shadow_root_get_put_7 [simp]:*
"get$_{ShadowRoot}$ *document_ptr (put*$_{Document}$ *shadow_root_ptr f h) = get*$_{ShadowRoot}$ *document_ptr h"*
⟨*proof*⟩

lemma *known_ptrs_implies: "DocumentClass.known_ptrs h ⟹ ShadowRootClass.known_ptrs h"*
⟨*proof*⟩

definition *delete*$_{ShadowRoot}$ *:: "(_) shadow_root_ptr ⇒ (_) heap ⇒ (_) heap option"* **where**
"delete$_{ShadowRoot}$ *shadow_root_ptr = delete*$_{Object}$ *(cast shadow_root_ptr)"*

lemma *delete*$_{ShadowRoot}$*_pointer_removed:*
assumes *"delete*$_{ShadowRoot}$ *ptr h = Some h'"*
shows *"ptr |∉| shadow_root_ptr_kinds h'"*
⟨*proof*⟩

lemma *delete*$_{ShadowRoot}$*_pointer_ptr_in_heap:*
assumes *"delete*$_{ShadowRoot}$ *ptr h = Some h'"*
shows *"ptr |∈| shadow_root_ptr_kinds h"*
⟨*proof*⟩

lemma *delete*$_{ShadowRoot}$*_ok:*
assumes *"ptr |∈| shadow_root_ptr_kinds h"*
shows *"delete*$_{ShadowRoot}$ *ptr h ≠ None"*
⟨*proof*⟩

lemma *shadow_root_delete_get_1 [simp]:*
"delete$_{ShadowRoot}$ *shadow_root_ptr h = Some h' ⟹ get*$_{ShadowRoot}$ *shadow_root_ptr h' = None"*
⟨*proof*⟩

lemma *shadow_root_different_delete_get [simp]:*
"shadow_root_ptr ≠ shadow_root_ptr' ⟹ delete$_{ShadowRoot}$ shadow_root_ptr' h = Some h' ⟹ get$_{ShadowRoot}$ shadow_root_ptr h' = get$_{ShadowRoot}$ shadow_root_ptr h"
⟨*proof*⟩

lemma *shadow_root_delete_get_2 [simp]:*
"delete$_{ShadowRoot}$ shadow_root_ptr h = Some h' ⟹ object_ptr ≠ cast shadow_root_ptr ⟹ get$_{Object}$ object_ptr h' = get$_{Object}$ object_ptr h"
⟨*proof*⟩
lemma *shadow_root_delete_get_3 [simp]:*
"delete$_{ShadowRoot}$ shadow_root_ptr h = Some h' ⟹ get$_{Node}$ node_ptr h' = get$_{Node}$ node_ptr h"
⟨*proof*⟩
lemma *shadow_root_delete_get_4 [simp]:*
"delete$_{ShadowRoot}$ shadow_root_ptr h = Some h' ⟹ get$_{Element}$ element_ptr h' = get$_{Element}$ element_ptr h"
⟨*proof*⟩
lemma *shadow_root_delete_get_5 [simp]:*
"delete$_{ShadowRoot}$ shadow_root_ptr h = Some h' ⟹ get$_{CharacterData}$ character_data_ptr h' = get$_{CharacterData}$ character_data_ptr h"
⟨*proof*⟩
lemma *shadow_root_delete_get_6 [simp]:*
"delete$_{ShadowRoot}$ shadow_root_ptr h = Some h' ⟹ get$_{Document}$ document_ptr h' = get$_{Document}$ document_ptr h"
⟨*proof*⟩
lemma *shadow_root_delete_get_7 [simp]:*
"delete$_{ShadowRoot}$ shadow_root_ptr h = Some h' ⟹ shadow_root_ptr' ≠ shadow_root_ptr ⟹ get$_{ShadowRoot}$ shadow_root_ptr' h' = get$_{ShadowRoot}$ shadow_root_ptr' h"
⟨*proof*⟩
end

2.2 Shadow Root Monad (ShadowRootMonad)

theory *ShadowRootMonad*
imports
"Core_DOM.DocumentMonad"
"../classes/ShadowRootClass"
begin

type_synonym *('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr, 'shadow_root_ptr, 'Object, 'Node, 'Element, 'CharacterData, 'Document, 'ShadowRoot, 'result) dom_prog = "((_) heap, exception, 'result) prog"*
register_default_tvars *"('object_ptr, 'node_ptr, 'element_ptr, 'character_data_ptr, 'document_ptr, 'shadow_root_ptr, 'Object, 'Node, 'Element, 'CharacterData, 'Document, 'ShadowRoot, 'result) dom_prog"*

global_interpretation *l_ptr_kinds_M shadow_root_ptr_kinds* **defines** *shadow_root_ptr_kinds_M = a_ptr_kinds_M*
⟨*proof*⟩
lemmas *shadow_root_ptr_kinds_M_defs = a_ptr_kinds_M_def*

lemma *shadow_root_ptr_kinds_M_eq:*
assumes *"|h ⊢ object_ptr_kinds_M|$_r$ = |h' ⊢ object_ptr_kinds_M|$_r$"*
shows *"|h ⊢ shadow_root_ptr_kinds_M|$_r$ = |h' ⊢ shadow_root_ptr_kinds_M|$_r$"*
⟨*proof*⟩

global_interpretation *l_dummy* **defines** *get_M$_{ShadowRoot}$ = "l_get_M.a_get_M get$_{ShadowRoot}$"* ⟨*proof*⟩
lemma *get_M_is_l_get_M: "l_get_M get$_{ShadowRoot}$ type_wf shadow_root_ptr_kinds"*
⟨*proof*⟩
lemmas *get_M_defs = get_M$_{ShadowRoot}$_def[unfolded l_get_M.a_get_M_def[OF get_M_is_l_get_M]]*

adhoc_overloading *get_M* ⇌ *get_M*$_{ShadowRoot}$

locale *l_get_M*$_{ShadowRoot}$*_lemmas = l_type_wf*$_{ShadowRoot}$
begin
sublocale *l_get_M*$_{CharacterData}$*_lemmas* ⟨*proof*⟩

interpretation *l_get_M get*$_{ShadowRoot}$ *type_wf shadow_root_ptr_kinds*
⟨*proof*⟩
lemmas *get_M*$_{ShadowRoot}$*_ok = get_M_ok[folded get_M*$_{ShadowRoot}$*_def]*
lemmas *get_M*$_{ShadowRoot}$*_ptr_in_heap = get_M_ptr_in_heap[folded get_M*$_{ShadowRoot}$*_def]*
end

global_interpretation *l_get_M*$_{ShadowRoot}$*_lemmas type_wf* ⟨*proof*⟩

global_interpretation *l_put_M type_wf shadow_root_ptr_kinds get*$_{ShadowRoot}$ *put*$_{ShadowRoot}$ **rewrites**
"a_get_M = get_M$_{ShadowRoot}$*"* **defines** *put_M*$_{ShadowRoot}$ *= a_put_M*
⟨*proof*⟩

lemmas *put_M_defs = a_put_M_def*
adhoc_overloading *put_M* ⇌ *put_M*$_{ShadowRoot}$

locale *l_put_M*$_{ShadowRoot}$*_lemmas = l_type_wf*$_{ShadowRoot}$
begin
sublocale *l_put_M*$_{CharacterData}$*_lemmas* ⟨*proof*⟩

interpretation *l_put_M type_wf shadow_root_ptr_kinds get*$_{ShadowRoot}$ *put*$_{ShadowRoot}$
⟨*proof*⟩
lemmas *put_M*$_{ShadowRoot}$*_ok = put_M_ok[folded put_M*$_{ShadowRoot}$*_def]*
end

global_interpretation *l_put_M*$_{ShadowRoot}$*_lemmas type_wf* ⟨*proof*⟩

lemma *shadow_root_put_get [simp]: "h ⊢ put_M*$_{ShadowRoot}$ *shadow_root_ptr setter v →*$_h$ *h'*
⟹ *(⋀x. getter (setter (λ_. v) x) = v)*
⟹ *h' ⊢ get_M*$_{ShadowRoot}$ *shadow_root_ptr getter →*$_r$ *v"*
⟨*proof*⟩
lemma *get_M_Mshadow_root_preserved1 [simp]:*
"shadow_root_ptr ≠ shadow_root_ptr'
⟹ *h ⊢ put_M*$_{ShadowRoot}$ *shadow_root_ptr setter v →*$_h$ *h'*
⟹ *preserved (get_M*$_{ShadowRoot}$ *shadow_root_ptr' getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_preserved [simp]:*
"h ⊢ put_M$_{ShadowRoot}$ *shadow_root_ptr setter v →*$_h$ *h'*
⟹ *(⋀x. getter (setter (λ_. v) x) = getter x)*
⟹ *preserved (get_M*$_{ShadowRoot}$ *shadow_root_ptr' getter) h h'"*
⟨*proof*⟩

lemma *get_M_Mshadow_root_preserved2 [simp]:*
"h ⊢ put_M$_{ShadowRoot}$ *shadow_root_ptr setter v →*$_h$ *h'* ⟹ *preserved (get_M*$_{Node}$ *node_ptr getter) h h'"*
⟨*proof*⟩

lemma *get_M_Mshadow_root_preserved3 [simp]:*
"cast shadow_root_ptr ≠ object_ptr
⟹ *h ⊢ put_M*$_{ShadowRoot}$ *shadow_root_ptr setter v →*$_h$ *h'*
⟹ *preserved (get_M*$_{Object}$ *object_ptr getter) h h'"*
⟨*proof*⟩
lemma *get_M_Mshadow_root_preserved4 [simp]:*
"h ⊢ put_M$_{ShadowRoot}$ *shadow_root_ptr setter v →*$_h$ *h'*
⟹ *(⋀x. getter (cast (setter (λ_. v) x)) = getter (cast x))*

```
  ⟹ preserved (get_M_Object object_ptr getter) h h'"
```
⟨proof⟩

lemma *get_M_Mshadow_root_preserved5 [simp]:*
```
  "cast shadow_root_ptr ≠ object_ptr
  ⟹ h ⊢ put_M_Object object_ptr setter v →_h h'
  ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter) h h'"
```
⟨proof⟩

lemma *get_M_Mshadow_root_preserved6 [simp]:*
```
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h' ⟹ preserved (get_M_Element element_ptr getter)
h h'"
```
⟨proof⟩
lemma *get_M_Mshadow_root_preserved7 [simp]:*
```
  "h ⊢ put_M_Element element_ptr setter v →_h h' ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter)
h h'"
```
⟨proof⟩
lemma *get_M_Mshadow_root_preserved8 [simp]:*
```
  "h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h'
    ⟹ preserved (get_M_CharacterData character_data_ptr getter) h h'"
```
⟨proof⟩
lemma *get_M_Mshadow_root_preserved9 [simp]:*
```
  "h ⊢ put_M_CharacterData character_data_ptr setter v →_h h'
    ⟹ preserved (get_M_ShadowRoot shadow_root_ptr getter) h h'"
```
⟨proof⟩
lemma *get_M_Mshadow_root_preserved10 [simp]:*
```
  "(⋀x. getter (cast (setter (λ_. v) x)) = getter (cast x))
    ⟹ h ⊢ put_M_ShadowRoot shadow_root_ptr setter v →_h h' ⟹ preserved (get_M_Object object_ptr getter)
h h'"
```
⟨proof⟩

lemma *new_element_get_M_ShadowRoot:*
```
  "h ⊢ new_element →_h h' ⟹ preserved (get_M_ShadowRoot ptr getter) h h'"
```
⟨proof⟩

lemma *new_character_data_get_M_ShadowRoot:*
```
  "h ⊢ new_character_data →_h h' ⟹ preserved (get_M_ShadowRoot ptr getter) h h'"
```
⟨proof⟩

lemma *new_document_get_M_ShadowRoot:*
```
  "h ⊢ new_document →_h h' ⟹ preserved (get_M_ShadowRoot ptr getter) h h'"
```
⟨proof⟩

definition *delete_ShadowRoot_M :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog"* **where**
```
  "delete_ShadowRoot_M shadow_root_ptr = do {
    h ← get_heap;
    (case delete_ShadowRoot shadow_root_ptr h of
      Some h ⇒ return_heap h |
      None ⇒ error HierarchyRequestError)
  }"
```
adhoc_overloading *delete_M ⇌ delete_ShadowRoot_M*

lemma *delete_ShadowRoot_M_ok [simp]:*
assumes *"shadow_root_ptr |∈| shadow_root_ptr_kinds h"*
shows *"h ⊢ ok (delete_ShadowRoot_M shadow_root_ptr)"*
⟨proof⟩

lemma *delete_ShadowRoot_M_ptr_in_heap:*
assumes *"h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'"*
shows *"shadow_root_ptr |∈| shadow_root_ptr_kinds h"*
⟨proof⟩

lemma *delete$_{ShadowRoot}$_M_ptr_not_in_heap:*
assumes *"h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h'"*
shows *"shadow_root_ptr |∉| shadow_root_ptr_kinds h'"*
⟨*proof*⟩

lemma *delete_shadow_root_pointers:*
assumes *"h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h'"*
shows *"object_ptr_kinds h = object_ptr_kinds h' |∪| {|cast shadow_root_ptr|}"*
⟨*proof*⟩

lemma *delete_shadow_root_get_M$_{Object}$: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h' ⟹ ptr ≠ cast shadow_root_ptr ⟹ preserved (get_M$_{Object}$ ptr getter) h h'"*
⟨*proof*⟩
lemma *delete_shadow_root_get_M$_{Node}$: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h' ⟹ preserved (get_M$_{Node}$ ptr getter) h h'"*
⟨*proof*⟩
lemma *delete_shadow_root_get_M$_{Element}$: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h' ⟹ preserved (get_M$_{Element}$ ptr getter) h h'"*
⟨*proof*⟩
lemma *delete_shadow_root_get_M$_{CharacterData}$: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h' ⟹ preserved (get_M$_{CharacterData}$ ptr getter) h h'"*
⟨*proof*⟩
lemma *delete_shadow_root_get_M$_{Document}$: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h' ⟹ preserved (get_M$_{Document}$ ptr getter) h h'"*
⟨*proof*⟩
lemma *delete_shadow_root_get_M$_{ShadowRoot}$: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h' ⟹ shadow_root_ptr ≠ shadow_root_ptr' ⟹ preserved (get_M$_{ShadowRoot}$ shadow_root_ptr' getter) h h'"*
⟨*proof*⟩

lemma *shadow_root_put_get_1 [simp]: "shadow_root_ptr ≠ shadow_root_ptr' ⟹ h ⊢ put_M$_{ShadowRoot}$ shadow_root_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{ShadowRoot}$ shadow_root_ptr' getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_2 [simp]: "(⋀x. getter (setter (λ_. v) x) = getter x) ⟹ h ⊢ put_M$_{ShadowRoot}$ shadow_root_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{ShadowRoot}$ shadow_root_ptr' getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_3 [simp]: "h ⊢ put_M$_{ShadowRoot}$ shadow_root_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{Element}$ element_ptr getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_4 [simp]: "h ⊢ put_M$_{Element}$ element_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{ShadowRoot}$ shadow_root_ptr getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_5 [simp]: "h ⊢ put_M$_{ShadowRoot}$ shadow_root_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{CharacterData}$ character_data_ptr getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_6 [simp]: "h ⊢ put_M$_{CharacterData}$ character_data_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{ShadowRoot}$ shadow_root_ptr getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_7 [simp]: "h ⊢ put_M$_{ShadowRoot}$ shadow_root_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{Document}$ document_ptr getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_8 [simp]: "h ⊢ put_M$_{Document}$ document_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{ShadowRoot}$ shadow_root_ptr getter) h h'"*
⟨*proof*⟩
lemma *shadow_root_put_get_9 [simp]: "(⋀x. getter (cast (setter (λ_. v) x)) = getter (cast x)) ⟹ h ⊢ put_M$_{ShadowRoot}$ shadow_root_ptr setter v →$_h$ h' ⟹ preserved (get_M$_{Object}$ object_ptr getter) h h'"*
⟨*proof*⟩

2.2.1 new_M

definition *new$_{ShadowRoot}$_M :: "(_, (_) shadow_root_ptr) dom_prog"*

where
"new$_{ShadowRoot}$_M = do {
h ← get_heap;
(new_ptr, h') ← return (new$_{ShadowRoot}$ h);
return_heap h';
return new_ptr
}"

lemma new$_{ShadowRoot}$_M_ok [simp]:
"h ⊢ ok new$_{ShadowRoot}$_M"
⟨*proof*⟩

lemma new$_{ShadowRoot}$_M_ptr_in_heap:
assumes "h ⊢ new$_{ShadowRoot}$_M →$_h$ h'"
and "h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr"
shows "new_shadow_root_ptr |∈| shadow_root_ptr_kinds h'"
⟨*proof*⟩

lemma new$_{ShadowRoot}$_M_ptr_not_in_heap:
assumes "h ⊢ new$_{ShadowRoot}$_M →$_h$ h'"
and "h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr"
shows "new_shadow_root_ptr |∉| shadow_root_ptr_kinds h"
⟨*proof*⟩

lemma new$_{ShadowRoot}$_M_new_ptr:
assumes "h ⊢ new$_{ShadowRoot}$_M →$_h$ h'"
and "h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr"
shows "object_ptr_kinds h' = object_ptr_kinds h |∪| {|cast new_shadow_root_ptr|}"
⟨*proof*⟩

lemma new$_{ShadowRoot}$_M_is_shadow_root_ptr:
assumes "h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr"
shows "is_shadow_root_ptr new_shadow_root_ptr"
⟨*proof*⟩

lemma new_shadow_root_mode:
assumes "h ⊢ new$_{ShadowRoot}$_M →$_h$ h'"
assumes "h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr"
shows "h' ⊢ get_M new_shadow_root_ptr mode →$_r$ Open"
⟨*proof*⟩

lemma new_shadow_root_children:
assumes "h ⊢ new$_{ShadowRoot}$_M →$_h$ h'"
assumes "h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr"
shows "h' ⊢ get_M new_shadow_root_ptr child_nodes →$_r$ []"
⟨*proof*⟩

lemma new_shadow_root_get_M$_{Object}$:
"h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ ptr ≠ cast new_shadow_root_ptr ⟹ preserved (get_M$_{Object}$ ptr getter) h h'"
⟨*proof*⟩
lemma new_shadow_root_get_M$_{Node}$:
"h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ preserved (get_M$_{Node}$ ptr getter) h h'"
⟨*proof*⟩
lemma new_shadow_root_get_M$_{Element}$:
"h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ preserved (get_M$_{Element}$ ptr getter) h h'"
⟨*proof*⟩
lemma new_shadow_root_get_M$_{CharacterData}$:
"h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ preserved (get_M$_{CharacterData}$ ptr getter) h h'"
⟨*proof*⟩

lemma *new_shadow_root_get_M$_{Document}$:*
"h ⊢ new$_{ShadowRoot}$_M →$_h$ h'
⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ preserved (get_M$_{Document}$ ptr getter) h h'"
⟨*proof*⟩
lemma *new_shadow_root_get_M$_{ShadowRoot}$:*
"h ⊢ new$_{ShadowRoot}$_M →$_h$ h'
⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr ⟹ ptr ≠ new_shadow_root_ptr
⟹ preserved (get_M$_{ShadowRoot}$ ptr getter) h h'"
⟨*proof*⟩

2.2.2 modified heaps

lemma *shadow_root_get_put_1 [simp]: "get$_{ShadowRoot}$ shadow_root_ptr (put$_{Object}$ ptr obj h) = (if ptr = cast shadow_root_ptr*
then cast obj else get shadow_root_ptr h)"
⟨*proof*⟩

lemma *shadow_root_ptr_kinds_new[simp]: "shadow_root_ptr_kinds (put$_{Object}$ ptr obj h) = shadow_root_ptr_kinds h |∪|*
(if is_shadow_root_ptr_kind ptr then {|the (cast ptr)|} else {||})"
⟨*proof*⟩

lemma *type_wf_put_I:*
assumes *"type_wf h"*
assumes *"DocumentClass.type_wf (put$_{Object}$ ptr obj h)"*
assumes *"is_shadow_root_ptr_kind ptr ⟹ is_shadow_root_kind obj"*
shows *"type_wf (put$_{Object}$ ptr obj h)"*
⟨*proof*⟩

lemma *type_wf_put_ptr_not_in_heap_E:*
assumes *"type_wf (put$_{Object}$ ptr obj h)"*
assumes *"ptr |∉| object_ptr_kinds h"*
shows *"type_wf h"*
⟨*proof*⟩

lemma *type_wf_put_ptr_in_heap_E:*
assumes *"type_wf (put$_{Object}$ ptr obj h)"*
assumes *"ptr |∈| object_ptr_kinds h"*
assumes *"DocumentClass.type_wf h"*
assumes *"is_shadow_root_ptr_kind ptr ⟹ is_shadow_root_kind (the (get ptr h))"*
shows *"type_wf h"*
⟨*proof*⟩

2.2.3 type_wf

lemma *new_element_type_wf_preserved [simp]:*
assumes *"h ⊢ new_element →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩

lemma *put_M$_{Element}$_tag_name_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M element_ptr tag_name_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩
lemma *put_M$_{Element}$_child_nodes_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M element_ptr RElement.child_nodes_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩
lemma *put_M$_{Element}$_attrs_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M element_ptr attrs_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩

lemma *put_M$_{Element}$_shadow_root_opt_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M element_ptr shadow_root_opt_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩

lemma *new_character_data_type_wf_preserved [simp]:*
assumes *"h ⊢ new_character_data →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩
lemma *put_M$_{CharacterData}$_val_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M character_data_ptr val_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩

lemma *new_document_type_wf_preserved [simp]:*
assumes *"h ⊢ new_document →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩

lemma *put_M$_{Document}$_doctype_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M document_ptr doctype_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩
lemma *put_M$_{Document}$_document_element_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M document_ptr document_element_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩
lemma *put_M$_{Document}$_disconnected_nodes_type_wf_preserved [simp]:*
assumes *"h ⊢ put_M document_ptr disconnected_nodes_update v →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩

lemma *put_M$_{ShadowRoot}$_mode_type_wf_preserved [simp]:*
"h ⊢ put_M shadow_root_ptr mode_update v →$_h$ h' ⟹ type_wf h = type_wf h'"
⟨*proof*⟩

lemma *put_M$_{ShadowRoot}$_child_nodes_type_wf_preserved [simp]:*
"h ⊢ put_M shadow_root_ptr RShadowRoot.child_nodes_update v →$_h$ h' ⟹ type_wf h = type_wf h'"
⟨*proof*⟩

lemma *shadow_root_ptr_kinds_small:*
assumes *"⋀object_ptr. preserved (get_M$_{Object}$ object_ptr RObject.nothing) h h'"*
shows *"shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"*
⟨*proof*⟩

lemma *shadow_root_ptr_kinds_preserved:*
assumes *"writes SW setter h h'"*
assumes *"h ⊢ setter →$_h$ h'"*
assumes *"⋀h h'. ∀w ∈ SW. h ⊢ w →$_h$ h' ⟶*
(∀object_ptr. preserved (get_M$_{Object}$ object_ptr RObject.nothing) h h')"
shows *"shadow_root_ptr_kinds h = shadow_root_ptr_kinds h'"*
⟨*proof*⟩

lemma *new_shadow_root_known_ptr:*
assumes *"h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr"*
shows *"known_ptr (cast new_shadow_root_ptr)"*
⟨*proof*⟩

lemma *new_shadow_root_type_wf_preserved [simp]: "h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ type_wf h = type_wf h'"*
⟨*proof*⟩

locale *l_new_shadow_root = l_type_wf +*
  assumes *new_shadow_root_types_preserved: "h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ type_wf h = type_wf h'"*

lemma *new_shadow_root_is_l_new_shadow_root [instances]: "l_new_shadow_root type_wf"*
  ⟨*proof*⟩

lemma *type_wf_preserved_small:*
  assumes *"⋀object_ptr. preserved (get_M$_{Object}$ object_ptr RObject.nothing) h h'"*
  assumes *"⋀node_ptr. preserved (get_M$_{Node}$ node_ptr RNode.nothing) h h'"*
  assumes *"⋀element_ptr. preserved (get_M$_{Element}$ element_ptr RElement.nothing) h h'"*
  assumes *"⋀character_data_ptr. preserved (get_M$_{CharacterData}$ character_data_ptr RCharacterData.nothing) h h'"*
  assumes *"⋀document_ptr. preserved (get_M$_{Document}$ document_ptr RDocument.nothing) h h'"*
  assumes *"⋀shadow_root_ptr. preserved (get_M$_{ShadowRoot}$ shadow_root_ptr RShadowRoot.nothing) h h'"*
  shows *"type_wf h = type_wf h'"*
  ⟨*proof*⟩

lemma *new_element_is_l_new_element [instances]:*
  "l_new_element type_wf"
  ⟨*proof*⟩

lemma *new_character_data_is_l_new_character_data [instances]:*
  "l_new_character_data type_wf"
  ⟨*proof*⟩

lemma *new_document_is_l_new_document [instances]:*
  "l_new_document type_wf"
  ⟨*proof*⟩

lemma *type_wf_preserved:*
  assumes *"writes SW setter h h'"*
  assumes *"h ⊢ setter →$_h$ h'"*
  assumes *"⋀h h' w. w ∈ SW ⟹ h ⊢ w →$_h$ h' ⟹*
∀object_ptr. preserved (get_M$_{Object}$ object_ptr RObject.nothing) h h'"
  assumes *"⋀h h' w. w ∈ SW ⟹ h ⊢ w →$_h$ h' ⟹*
∀node_ptr. preserved (get_M$_{Node}$ node_ptr RNode.nothing) h h'"
  assumes *"⋀h h' w. w ∈ SW ⟹ h ⊢ w →$_h$ h' ⟹*
∀element_ptr. preserved (get_M$_{Element}$ element_ptr RElement.nothing) h h'"
  assumes *"⋀h h' w. w ∈ SW ⟹ h ⊢ w →$_h$ h' ⟹*
∀character_data_ptr. preserved (get_M$_{CharacterData}$ character_data_ptr RCharacterData.nothing) h h'"
  assumes *"⋀h h' w. w ∈ SW ⟹ h ⊢ w →$_h$ h' ⟹*
∀document_ptr. preserved (get_M$_{Document}$ document_ptr RDocument.nothing) h h'"
  assumes *"⋀h h' w. w ∈ SW ⟹ h ⊢ w →$_h$ h' ⟹*
∀shadow_root_ptr. preserved (get_M$_{ShadowRoot}$ shadow_root_ptr RShadowRoot.nothing) h h'"
  shows *"type_wf h = type_wf h'"*
⟨*proof*⟩

lemma *type_wf_drop: "type_wf h ⟹ type_wf (Heap (fmdrop ptr (the_heap h)))"*
  ⟨*proof*⟩

lemma *delete_shadow_root_type_wf_preserved [simp]:*
  assumes *"h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h'"*
  assumes *"type_wf h"*
  shows *"type_wf h'"*
  ⟨*proof*⟩
end

2.3 The Shadow DOM (Shadow_DOM)

theory *Shadow_DOM*
  imports
    "monads/ShadowRootMonad"

```
    Core_DOM.Core_DOM
begin

abbreviation "safe_shadow_root_element_types ≡ {''article'', ''aside'', ''blockquote'', ''body'',
  ''div'', ''footer'', ''h1'', ''h2'', ''h3'', ''h4'', ''h5'', ''h6'', ''header'', ''main'',
  ''nav'', ''p'', ''section'', ''span''}"
```

2.3.1 Function Definitions

get_child_nodes

```
locale l_get_child_nodes_Shadow_DOM_defs =
  CD: l_get_child_nodes_Core_DOM_defs
begin
definition get_child_nodes_shadow_root_ptr :: "(_) shadow_root_ptr ⇒ unit
    ⇒ (_, (_) node_ptr list) dom_prog" where
  "get_child_nodes_shadow_root_ptr shadow_root_ptr _ = get_M shadow_root_ptr RShadowRoot.child_nodes"

definition a_get_child_nodes_tups :: "(((_) object_ptr ⇒ bool) × ((_) object_ptr ⇒ unit
    ⇒ (_, (_) node_ptr list) dom_prog)) list" where
  "a_get_child_nodes_tups ≡ [(is_shadow_root_ptr_object_ptr, get_child_nodes_shadow_root_ptr ∘ the ∘ cast)]"

definition a_get_child_nodes :: "(_) object_ptr ⇒ (_, (_) node_ptr list) dom_prog" where
  "a_get_child_nodes ptr = invoke (CD.a_get_child_nodes_tups @ a_get_child_nodes_tups) ptr ()"

definition a_get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" where
  "a_get_child_nodes_locs ptr ≡
    (if is_shadow_root_ptr_kind ptr
    then {preserved (get_M (the (cast ptr)) RShadowRoot.child_nodes)} else {}) ∪
    CD.a_get_child_nodes_locs ptr"

definition first_child :: "(_) object_ptr ⇒ (_, (_) node_ptr option) dom_prog"
  where
    "first_child ptr = do {
      children ← a_get_child_nodes ptr;
      return (case children of [] ⇒ None | child#_ ⇒ Some child)}"
end

global_interpretation l_get_child_nodes_Shadow_DOM_defs defines
  get_child_nodes = l_get_child_nodes_Shadow_DOM_defs.a_get_child_nodes and
  get_child_nodes_locs = l_get_child_nodes_Shadow_DOM_defs.a_get_child_nodes_locs
  ⟨proof⟩

locale l_get_child_nodes_Shadow_DOM =
  l_type_wf type_wf +
  l_known_ptr known_ptr +
  l_get_child_nodes_Shadow_DOM_defs +
  l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
  CD: l_get_child_nodes_Core_DOM type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes_Core_DOM
  get_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and get_child_nodes :: "(_) object_ptr ⇒ (_, (_) node_ptr list) dom_prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_, (_) node_ptr list) dom_prog"
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes known_ptr_impl: "known_ptr = ShadowRootClass.known_ptr"
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes get_child_nodes_impl: "get_child_nodes = a_get_child_nodes"
  assumes get_child_nodes_locs_impl: "get_child_nodes_locs = a_get_child_nodes_locs"
begin
```

lemmas *get_child_nodes_def = get_child_nodes_impl[unfolded a_get_child_nodes_def get_child_nodes_def]*
lemmas *get_child_nodes_locs_def = get_child_nodes_locs_impl[unfolded a_get_child_nodes_locs_def get_child_nodes_locs_def, folded CD.get_child_nodes_locs_impl]*

lemma *get_child_nodes_ok:*
assumes *"known_ptr ptr"*
assumes *"type_wf h"*
assumes *"ptr |∈| object_ptr_kinds h"*
shows *"h ⊢ ok (get_child_nodes ptr)"*
⟨*proof*⟩

lemma *get_child_nodes_ptr_in_heap:*
assumes *"h ⊢ get_child_nodes ptr →r children"*
shows *"ptr |∈| object_ptr_kinds h"*
⟨*proof*⟩

lemma *get_child_nodes_pure [simp]:*
"pure (get_child_nodes ptr) h"
⟨*proof*⟩

lemma *get_child_nodes_reads: "reads (get_child_nodes_locs ptr) (get_child_nodes ptr) h h'"*
⟨*proof*⟩
end

interpretation *i_get_child_nodes?: l_get_child_nodes$_{Shadow_DOM}$ type_wf known_ptr DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs*
⟨*proof*⟩
declare *l_get_child_nodes$_{Shadow_DOM}$_axioms [instances]*

lemma *get_child_nodes_is_l_get_child_nodes [instances]: "l_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs"*
⟨*proof*⟩

new_document **locale** *l_new_document_get_child_nodes$_{Shadow_DOM}$ =*
CD: l_new_document_get_child_nodes$_{Core_DOM}$ type_wf$_{Core_DOM}$ known_ptr$_{Core_DOM}$ get_child_nodes$_{Core_DOM}$ get_child_nodes_locs$_{Core_DOM}$
+ l_get_child_nodes$_{Shadow_DOM}$ type_wf known_ptr type_wf$_{Core_DOM}$ known_ptr$_{Core_DOM}$ get_child_nodes get_child_nodes_locs get_child_nodes$_{Core_DOM}$ get_child_nodes_locs$_{Core_DOM}$
for *type_wf :: "(_) heap ⇒ bool"*
and *known_ptr :: "(_) object_ptr ⇒ bool"*
and *get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"*
and *get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
and *type_wf$_{Core_DOM}$ :: "(_) heap ⇒ bool"*
and *known_ptr$_{Core_DOM}$ :: "(_) object_ptr ⇒ bool"*
and *get_child_nodes$_{Core_DOM}$ :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"*
and *get_child_nodes_locs$_{Core_DOM}$ :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
begin
lemma *get_child_nodes_new_document:*
"ptr' ≠ cast new_document_ptr ⟹ h ⊢ new_document →r new_document_ptr ⟹ h ⊢ new_document →h h' ⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
⟨*proof*⟩

lemma *new_document_no_child_nodes:*
"h ⊢ new_document →r new_document_ptr ⟹ h ⊢ new_document →h h' ⟹ h' ⊢ get_child_nodes (cast new_document_ptr) →r []"
⟨*proof*⟩
end
interpretation *i_new_document_get_child_nodes?:*
l_new_document_get_child_nodes$_{Shadow_DOM}$ type_wf known_ptr get_child_nodes get_child_nodes_locs DocumentClass.type_wf DocumentClass.known_ptr Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs
⟨*proof*⟩

declare *l_new_document_get_child_nodes$_{Core_DOM}$_axioms[instances]*

lemma *new_document_get_child_nodes_is_l_new_document_get_child_nodes [instances]:*
"l_new_document_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs"
⟨*proof*⟩

new_shadow_root **locale** *l_new_shadow_root_get_child_nodes$_{Shadow_DOM}$ =*
l_get_child_nodes$_{Shadow_DOM}$ type_wf known_ptr type_wf$_{Core_DOM}$ known_ptr$_{Core_DOM}$ get_child_nodes
get_child_nodes_locs get_child_nodes$_{Core_DOM}$ get_child_nodes_locs$_{Core_DOM}$
for *type_wf :: "(_) heap ⇒ bool"*
and *known_ptr :: "(_) object_ptr ⇒ bool"*
and *get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"*
and *get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
and *type_wf$_{Core_DOM}$ :: "(_) heap ⇒ bool"*
and *known_ptr$_{Core_DOM}$ :: "(_) object_ptr ⇒ bool"*
and *get_child_nodes$_{Core_DOM}$ :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"*
and *get_child_nodes_locs$_{Core_DOM}$ :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
begin
lemma *get_child_nodes_new_shadow_root:*
"ptr' ≠ cast new_shadow_root_ptr ⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
⟨*proof*⟩

lemma *new_shadow_root_no_child_nodes:*
"h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr ⟹ h ⊢ new$_{ShadowRoot}$_M →$_h$ h'
⟹ h' ⊢ get_child_nodes (cast new_shadow_root_ptr) →$_r$ []"
⟨*proof*⟩
end
interpretation *i_new_shadow_root_get_child_nodes?:*
l_new_shadow_root_get_child_nodes$_{Shadow_DOM}$ type_wf known_ptr get_child_nodes get_child_nodes_locs
DocumentClass.type_wf DocumentClass.known_ptr Core_DOM_Functions.get_child_nodes
Core_DOM_Functions.get_child_nodes_locs
⟨*proof*⟩
declare *l_new_shadow_root_get_child_nodes$_{Shadow_DOM}$_def[instances]*

locale *l_new_shadow_root_get_child_nodes = l_get_child_nodes +*
assumes *get_child_nodes_new_shadow_root:*
"ptr' ≠ cast new_shadow_root_ptr ⟹ h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
assumes *new_shadow_root_no_child_nodes:*
"h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr ⟹ h ⊢ new$_{ShadowRoot}$_M →$_h$ h'
⟹ h' ⊢ get_child_nodes (cast new_shadow_root_ptr) →$_r$ []"

lemma *new_shadow_root_get_child_nodes_is_l_new_shadow_root_get_child_nodes [instances]:*
"l_new_shadow_root_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs"
⟨*proof*⟩

new_element **locale** *l_new_element_get_child_nodes$_{Shadow_DOM}$ =*
l_get_child_nodes$_{Shadow_DOM}$ +
l_new_element_get_child_nodes$_{Core_DOM}$ type_wf$_{Core_DOM}$ known_ptr$_{Core_DOM}$ get_child_nodes$_{Core_DOM}$
get_child_nodes_locs$_{Core_DOM}$
begin
lemma *get_child_nodes_new_element:*
"ptr' ≠ cast new_element_ptr ⟹ h ⊢ new_element →$_r$ new_element_ptr ⟹ h ⊢ new_element →$_h$ h'
⟹ r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
⟨*proof*⟩

lemma *new_element_no_child_nodes:*
"h ⊢ new_element →$_r$ new_element_ptr ⟹ h ⊢ new_element →$_h$ h'
⟹ h' ⊢ get_child_nodes (cast new_element_ptr) →$_r$ []"
⟨*proof*⟩
end

interpretation *i_new_element_get_child_nodes?:*
l_new_element_get_child_nodes$_{Shadow_DOM}$ *type_wf known_ptr DocumentClass.type_wf*
DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes
Core_DOM_Functions.get_child_nodes_locs
⟨*proof*⟩
declare *l_new_element_get_child_nodes*$_{Shadow_DOM}$*_axioms[instances]*

lemma *new_element_get_child_nodes_is_l_new_element_get_child_nodes [instances]:*
"l_new_element_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs"
⟨*proof*⟩

delete_shadow_root

locale *l_delete_shadow_root_get_child_nodes*$_{Shadow_DOM}$ =
l_get_child_nodes$_{Shadow_DOM}$
begin
lemma *get_child_nodes_delete_shadow_root:*
"ptr' ≠ cast shadow_root_ptr ⟹ h ⊢ delete$_{ShadowRoot}$*_M shadow_root_ptr →*$_h$ *h' ⟹*
r ∈ get_child_nodes_locs ptr' ⟹ r h h'"
⟨*proof*⟩
end

locale *l_delete_shadow_root_get_child_nodes = l_get_child_nodes_defs +*
assumes *get_child_nodes_delete_shadow_root:*
"ptr' ≠ cast shadow_root_ptr ⟹ h ⊢ delete$_{ShadowRoot}$*_M shadow_root_ptr →*$_h$ *h' ⟹*
r ∈ get_child_nodes_locs ptr' ⟹ r h h'"

interpretation *l_delete_shadow_root_get_child_nodes*$_{Shadow_DOM}$ *type_wf known_ptr DocumentClass.type_wf*
DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes
Core_DOM_Functions.get_child_nodes_locs
⟨*proof*⟩

lemma *l_delete_shadow_root_get_child_nodes_get_child_nodes_locs [instances]: "l_delete_shadow_root_get_child_node*
get_child_nodes_locs"
⟨*proof*⟩

set_child_nodes

locale *l_set_child_nodes*$_{Shadow_DOM}$*_defs =*
CD: l_set_child_nodes$_{Core_DOM}$*_defs*
begin
definition *set_child_nodes*$_{shadow_root_ptr}$ *:: "(_) shadow_root_ptr ⇒ (_) node_ptr list*
⇒ (_, unit) dom_prog" **where**
"set_child_nodes$_{shadow_root_ptr}$ *shadow_root_ptr = put_M shadow_root_ptr RShadowRoot.child_nodes_update"*

definition *a_set_child_nodes_tups :: "(((_) object_ptr ⇒ bool) × ((_) object_ptr ⇒ (_) node_ptr list*
⇒ (_, unit) dom_prog)) list" **where**
"a_set_child_nodes_tups ≡ [(is_shadow_root_ptr$_{object_ptr}$*, set_child_nodes*$_{shadow_root_ptr}$ *∘ the ∘ cast)]"*

definition *a_set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ (_, unit) dom_prog"* **where**
"a_set_child_nodes ptr children = invoke (CD.a_set_child_nodes_tups @ a_set_child_nodes_tups)
ptr children"

definition *a_set_child_nodes_locs :: "(_) object_ptr ⇒ (_, unit) dom_prog set"*
where
"a_set_child_nodes_locs ptr ≡
(if is_shadow_root_ptr_kind ptr
then all_args (put_M (the (cast ptr)) RShadowRoot.child_nodes_update)
else {}) ∪ CD.a_set_child_nodes_locs ptr"
end

global_interpretation *l_set_child_nodes*$_{Shadow_DOM}$*_defs* **defines**
set_child_nodes = l_set_child_nodes$_{Shadow_DOM}$*_defs.a_set_child_nodes* **and**

```
  set_child_nodes_locs = l_set_child_nodes_Shadow_DOM_defs.a_set_child_nodes_locs
  ⟨proof⟩

locale l_set_child_nodes_Shadow_DOM =
  l_type_wf type_wf +
  l_known_ptr known_ptr +
  l_set_child_nodes_Shadow_DOM_defs +
  l_set_child_nodes_defs set_child_nodes set_child_nodes_locs +
  CD: l_set_child_nodes_Core_DOM type_wf_Core_DOM known_ptr_Core_DOM set_child_nodes_Core_DOM
  set_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ (_, unit) dom_prog"
    and set_child_nodes_locs :: "(_) object_ptr ⇒ (_, unit) dom_prog set"
    and set_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ (_, unit) dom_prog"
    and set_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ (_, unit) dom_prog set" +
  assumes known_ptr_impl: "known_ptr = ShadowRootClass.known_ptr"
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes set_child_nodes_impl: "set_child_nodes = a_set_child_nodes"
  assumes set_child_nodes_locs_impl: "set_child_nodes_locs = a_set_child_nodes_locs"
begin
lemmas set_child_nodes_def = set_child_nodes_impl[unfolded a_set_child_nodes_def set_child_nodes_def]
lemmas set_child_nodes_locs_def = set_child_nodes_locs_impl[unfolded a_set_child_nodes_locs_def
    set_child_nodes_locs_def, folded CD.set_child_nodes_locs_impl]

lemma set_child_nodes_writes: "writes (set_child_nodes_locs ptr) (set_child_nodes ptr children) h h'"
  ⟨proof⟩

lemma set_child_nodes_pointers_preserved:
  assumes "w ∈ set_child_nodes_locs object_ptr"
  assumes "h ⊢ w →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  ⟨proof⟩

lemma set_child_nodes_types_preserved:
  assumes "w ∈ set_child_nodes_locs object_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  ⟨proof⟩
end

interpretation
  i_set_child_nodes?: l_set_child_nodes_Shadow_DOM type_wf known_ptr DocumentClass.type_wf
  DocumentClass.known_ptr set_child_nodes set_child_nodes_locs Core_DOM_Functions.set_child_nodes
  Core_DOM_Functions.set_child_nodes_locs
  ⟨proof⟩
declare l_set_child_nodes_Shadow_DOM_axioms[instances]

lemma set_child_nodes_is_l_set_child_nodes [instances]: "l_set_child_nodes type_wf set_child_nodes
    set_child_nodes_locs"
  ⟨proof⟩
```

get_child_nodes `locale l_set_child_nodes_get_child_nodes_Shadow_DOM =`

```
  l_get_child_nodes_Shadow_DOM
  type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes get_child_nodes_locs
  get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
  + l_set_child_nodes_Shadow_DOM
  type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM set_child_nodes set_child_nodes_locs
  set_child_nodes_Core_DOM set_child_nodes_locs_Core_DOM
  + CD: l_set_child_nodes_get_child_nodes_Core_DOM
  type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
```

```
  set_child_nodes_Core_DOM set_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    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_child_nodes_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit) prog"
    and set_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
begin

lemma set_child_nodes_get_child_nodes:
  assumes "known_ptr ptr"
  assumes "type_wf h"
  assumes "h ⊢ set_child_nodes ptr children →_h h'"
  shows "h' ⊢ get_child_nodes ptr →_r children"
⟨proof⟩

lemma set_child_nodes_get_child_nodes_different_pointers:
  assumes "ptr ≠ ptr'"
  assumes "w ∈ set_child_nodes_locs ptr"
  assumes "h ⊢ w →_h h'"
  assumes "r ∈ get_child_nodes_locs ptr'"
  shows "r h h'"
  ⟨proof⟩

end

interpretation
  i_set_child_nodes_get_child_nodes?: l_set_child_nodes_get_child_nodes_Shadow_DOM type_wf known_ptr
  DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes get_child_nodes_locs
  Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs set_child_nodes
  set_child_nodes_locs Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs
  ⟨proof⟩
declare l_set_child_nodes_get_child_nodes_Shadow_DOM_axioms[instances]

lemma set_child_nodes_get_child_nodes_is_l_set_child_nodes_get_child_nodes [instances]:
  "l_set_child_nodes_get_child_nodes type_wf known_ptr get_child_nodes get_child_nodes_locs
    set_child_nodes set_child_nodes_locs"
  ⟨proof⟩
```

set_tag_type

```
locale l_set_tag_name_Shadow_DOM =
  CD: l_set_tag_name_Core_DOM type_wf_Core_DOM set_tag_name set_tag_name_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and set_tag_name :: "(_) element_ptr ⇒ tag_name ⇒ (_, unit) dom_prog"
    and set_tag_name_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin
lemmas set_tag_name_def = CD.set_tag_name_impl[unfolded CD.a_set_tag_name_def set_tag_name_def]
lemmas set_tag_name_locs_def = CD.set_tag_name_locs_impl[unfolded CD.a_set_tag_name_locs_def
    set_tag_name_locs_def]

lemma set_tag_name_ok:
  "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (set_tag_name element_ptr tag)"
```

⟨*proof*⟩

lemma *set_tag_name_writes:*
"writes (set_tag_name_locs element_ptr) (set_tag_name element_ptr tag) h h'"
⟨*proof*⟩

lemma *set_tag_name_pointers_preserved:*
assumes *"w ∈ set_tag_name_locs element_ptr"*
assumes *"h ⊢ w →$_h$ h'"*
shows *"object_ptr_kinds h = object_ptr_kinds h'"*
⟨*proof*⟩

lemma *set_tag_name_typess_preserved:*
assumes *"w ∈ set_tag_name_locs element_ptr"*
assumes *"h ⊢ w →$_h$ h'"*
shows *"type_wf h = type_wf h'"*
⟨*proof*⟩
end

interpretation *i_set_tag_name?: l_set_tag_name$_{Shadow_DOM}$ type_wf DocumentClass.type_wf set_tag_name set_tag_name_locs*
⟨*proof*⟩
declare *l_set_tag_name$_{Shadow_DOM}$_axioms [instances]*

lemma *set_tag_name_is_l_set_tag_name [instances]:*
"l_set_tag_name type_wf set_tag_name set_tag_name_locs"
⟨*proof*⟩

get_child_nodes **locale** *l_set_tag_name_get_child_nodes$_{Shadow_DOM}$ =*
l_set_tag_name$_{Shadow_DOM}$ +
l_get_child_nodes$_{Shadow_DOM}$ +
CD: l_set_tag_name_get_child_nodes$_{Core_DOM}$ type_wf$_{Core_DOM}$ set_tag_name set_tag_name_locs known_ptr$_{Core_DOM}$ get_child_nodes$_{Core_DOM}$ get_child_nodes_locs$_{Core_DOM}$
begin
lemma *set_tag_name_get_child_nodes:*
"∀w ∈ set_tag_name_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
⟨*proof*⟩
end

interpretation
i_set_tag_name_get_child_nodes?: l_set_tag_name_get_child_nodes$_{Shadow_DOM}$ type_wf DocumentClass.type_wf set_tag_name set_tag_name_locs known_ptr DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs
⟨*proof*⟩
declare *l_set_tag_name_get_child_nodes$_{Shadow_DOM}$_axioms[instances]*

lemma *set_tag_name_get_child_nodes_is_l_set_tag_name_get_child_nodes [instances]:*
"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"
⟨*proof*⟩

get_shadow_root

locale *l_get_shadow_root$_{Shadow_DOM}$_defs*
begin
definition *a_get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"*
where
"a_get_shadow_root element_ptr = get_M element_ptr shadow_root_opt"

definition *a_get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
where
"a_get_shadow_root_locs element_ptr ≡ {preserved (get_M element_ptr shadow_root_opt)}"

end

global_interpretation *l_get_shadow_root*$_{Shadow_DOM}$*_defs*
defines *get_shadow_root = a_get_shadow_root*
and *get_shadow_root_locs = a_get_shadow_root_locs*
⟨*proof*⟩

locale *l_get_shadow_root_defs =*
fixes *get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"*
fixes *get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*

locale *l_get_shadow_root*$_{Shadow_DOM}$ *=*
l_get_shadow_root$_{Shadow_DOM}$*_defs +*
l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
l_type_wf type_wf
for *type_wf :: "(_) heap ⇒ bool"*
and *get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"*
and *get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +*
assumes *type_wf_impl: "type_wf = ShadowRootClass.type_wf"*
assumes *get_shadow_root_impl: "get_shadow_root = a_get_shadow_root"*
assumes *get_shadow_root_locs_impl: "get_shadow_root_locs = a_get_shadow_root_locs"*
begin
lemmas *get_shadow_root_def = get_shadow_root_impl[unfolded get_shadow_root_def a_get_shadow_root_def]*
lemmas *get_shadow_root_locs_def = get_shadow_root_locs_impl[unfolded get_shadow_root_locs_def*
a_get_shadow_root_locs_def]

lemma *get_shadow_root_ok:*
"type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (get_shadow_root element_ptr)"
⟨*proof*⟩

lemma *get_shadow_root_pure [simp]: "pure (get_shadow_root element_ptr) h"*
⟨*proof*⟩

lemma *get_shadow_root_ptr_in_heap:*
assumes *"h ⊢ get_shadow_root element_ptr →$_r$ children"*
shows *"element_ptr |∈| element_ptr_kinds h"*
⟨*proof*⟩

lemma *get_shadow_root_reads:*
"reads (get_shadow_root_locs element_ptr) (get_shadow_root element_ptr) h h'"
⟨*proof*⟩
end

interpretation *i_get_shadow_root?: l_get_shadow_root*$_{Shadow_DOM}$ *type_wf get_shadow_root*
get_shadow_root_locs
⟨*proof*⟩
declare *l_get_shadow_root*$_{Shadow_DOM}$*_axioms [instances]*

locale *l_get_shadow_root = l_type_wf + l_get_shadow_root_defs +*
assumes *get_shadow_root_reads:*
"reads (get_shadow_root_locs element_ptr) (get_shadow_root element_ptr) h h'"
assumes *get_shadow_root_ok:*
"type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (get_shadow_root element_ptr)"
assumes *get_shadow_root_ptr_in_heap:*
"h ⊢ ok (get_shadow_root element_ptr) ⟹ element_ptr |∈| element_ptr_kinds h"
assumes *get_shadow_root_pure [simp]:*
"pure (get_shadow_root element_ptr) h"

lemma *get_shadow_root_is_l_get_shadow_root [instances]:*
"l_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
⟨*proof*⟩

set_disconnected_nodes **locale** *l_set_disconnected_nodes_get_shadow_root*$_{Shadow_DOM}$ *=*

```
  l_set_disconnected_nodes_Core_DOM type_wf_Core_DOM set_disconnected_nodes set_disconnected_nodes_locs
+
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    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 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 set_disconnected_nodes_get_shadow_root:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  ⟨proof⟩
end

locale l_set_disconnected_nodes_get_shadow_root =
  l_set_disconnected_nodes_defs +
  l_get_shadow_root_defs +
  assumes set_disconnected_nodes_get_shadow_root:
    "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation
  i_set_disconnected_nodes_get_shadow_root?: l_set_disconnected_nodes_get_shadow_root_Shadow_DOM
  type_wf DocumentClass.type_wf set_disconnected_nodes set_disconnected_nodes_locs get_shadow_root
  get_shadow_root_locs
  ⟨proof⟩
declare l_set_disconnected_nodes_get_shadow_root_Shadow_DOM_axioms[instances]

lemma set_disconnected_nodes_get_shadow_root_is_l_set_disconnected_nodes_get_shadow_root [instances]:
  "l_set_disconnected_nodes_get_shadow_root set_disconnected_nodes_locs get_shadow_root_locs"
  ⟨proof⟩
```

set_tag_type

```
locale l_set_tag_name_get_shadow_root_Core_DOM =
  l_set_tag_name_Shadow_DOM +
  l_get_shadow_root_Shadow_DOM
begin
lemma set_tag_name_get_shadow_root:
  "∀w ∈ set_tag_name_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  ⟨proof⟩
end

locale l_set_tag_name_get_shadow_root = l_set_tag_name + l_get_shadow_root +
  assumes set_tag_name_get_shadow_root:
    "∀w ∈ set_tag_name_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation
  i_set_tag_name_get_shadow_root?: l_set_tag_name_get_shadow_root_Core_DOM type_wf DocumentClass.type_wf
  set_tag_name set_tag_name_locs
  get_shadow_root get_shadow_root_locs
  ⟨proof⟩
declare l_set_tag_name_get_shadow_root_Core_DOM_axioms[instances]

lemma set_tag_name_get_shadow_root_is_l_set_tag_name_get_shadow_root [instances]:
  "l_set_tag_name_get_shadow_root type_wf set_tag_name set_tag_name_locs get_shadow_root
                                  get_shadow_root_locs"
  ⟨proof⟩
```

set_child_nodes

```
locale l_set_child_nodes_get_shadow_root_Shadow_DOM =
  l_set_child_nodes_Shadow_DOM type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM set_child_nodes
  set_child_nodes_locs set_child_nodes_Core_DOM set_child_nodes_locs_Core_DOM +
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
```

```
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    and set_child_nodes :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit) prog"
    and set_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog set"
    and set_child_nodes_Core_DOM :: "(_) object_ptr ⇒ (_) node_ptr list ⇒ ((_) heap, exception, unit)
prog"
    and set_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, unit) prog 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 set_child_nodes_get_shadow_root: "∀w ∈ set_child_nodes_locs ptr. (h ⊢ w →_h h' ⟶
  (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  ⟨proof⟩
end

locale l_set_child_nodes_get_shadow_root = l_set_child_nodes_defs + l_get_shadow_root_defs +
  assumes set_child_nodes_get_shadow_root:
    "∀w ∈ set_child_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation
  i_set_child_nodes_get_shadow_root?: l_set_child_nodes_get_shadow_root_Shadow_DOM type_wf known_ptr
  DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
  Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs get_shadow_root
  get_shadow_root_locs
  ⟨proof⟩
declare l_set_child_nodes_get_shadow_root_Shadow_DOM_axioms[instances]

lemma set_child_nodes_get_shadow_root_is_l_set_child_nodes_get_shadow_root [instances]:
  "l_set_child_nodes_get_shadow_root set_child_nodes_locs get_shadow_root_locs"
  ⟨proof⟩
```

delete_shadow_root locale `l_delete_shadow_root_get_shadow_root_Shadow_DOM =`
`l_get_shadow_root_Shadow_DOM`

```
begin
lemma get_shadow_root_delete_shadow_root:  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  ⟨proof⟩
end

locale l_delete_shadow_root_get_shadow_root = l_get_shadow_root_defs +
  assumes get_shadow_root_delete_shadow_root:  "h ⊢ delete_ShadowRoot_M shadow_root_ptr →_h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
interpretation l_delete_shadow_root_get_shadow_root_Shadow_DOM type_wf get_shadow_root
  get_shadow_root_locs
  ⟨proof⟩

lemma l_delete_shadow_root_get_shadow_root_get_shadow_root_locs [instances]: "l_delete_shadow_root_get_shadow_roo
get_shadow_root_locs"
  ⟨proof⟩
```

new_character_data locale `l_new_character_data_get_shadow_root_Shadow_DOM =`
`l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs`

```
  for type_wf :: "(_) heap ⇒ bool"
    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 get_shadow_root_new_character_data:
  "h ⊢ new_character_data →_r new_character_data_ptr ⟹ h ⊢ new_character_data →_h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  ⟨proof⟩
end
```

locale *l_new_character_data_get_shadow_root = l_new_character_data + l_get_shadow_root +*
  assumes *get_shadow_root_new_character_data:*
    "h ⊢ new_character_data →r new_character_data_ptr
        ⟹ h ⊢ new_character_data →h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"

interpretation *i_new_character_data_get_shadow_root?:*
  l_new_character_data_get_shadow_root$_{Shadow_DOM}$ *type_wf get_shadow_root get_shadow_root_locs*
  ⟨*proof*⟩
declare *l_new_character_data_get_shadow_root*$_{Shadow_DOM}$*_axioms [instances]*

lemma *new_character_data_get_shadow_root_is_l_new_character_data_get_shadow_root [instances]:*
  "l_new_character_data_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  ⟨*proof*⟩

new_document **locale** *l_new_document_get_shadow_root*$_{Shadow_DOM}$ *=*
  l_get_shadow_root$_{Shadow_DOM}$ *type_wf get_shadow_root get_shadow_root_locs*
  for *type_wf :: "(_) heap ⇒ bool"*
    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 *get_shadow_root_new_document:*
  "h ⊢ new_document →r new_document_ptr ⟹ h ⊢ new_document →h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  ⟨*proof*⟩
end

locale *l_new_document_get_shadow_root = l_new_document + l_get_shadow_root +*
  assumes *get_shadow_root_new_document:*
    "h ⊢ new_document →r new_document_ptr
        ⟹ h ⊢ new_document →h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"

interpretation *i_new_document_get_shadow_root?:*
  l_new_document_get_shadow_root$_{Shadow_DOM}$ *type_wf get_shadow_root get_shadow_root_locs*
  ⟨*proof*⟩
declare *l_new_document_get_shadow_root*$_{Shadow_DOM}$*_axioms [instances]*

lemma *new_document_get_shadow_root_is_l_new_document_get_shadow_root [instances]:*
  "l_new_document_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  ⟨*proof*⟩

new_element **locale** *l_new_element_get_shadow_root*$_{Shadow_DOM}$ *=*
  l_get_shadow_root$_{Shadow_DOM}$ *type_wf get_shadow_root get_shadow_root_locs*
  for *type_wf :: "(_) heap ⇒ bool"*
    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 *get_shadow_root_new_element:*
  "ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →r new_element_ptr ⟹ h ⊢ new_element →h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  ⟨*proof*⟩

lemma *new_element_no_shadow_root:*
  "h ⊢ new_element →r new_element_ptr ⟹ h ⊢ new_element →h h'
   ⟹ h' ⊢ get_shadow_root new_element_ptr →r None"
  ⟨*proof*⟩
end

locale *l_new_element_get_shadow_root = l_new_element + l_get_shadow_root +*
  assumes *get_shadow_root_new_element:*
    "ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →r new_element_ptr
        ⟹ h ⊢ new_element →h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"

```
  assumes new_element_no_shadow_root:
    "h ⊢ new_element →_r new_element_ptr ⟹ h ⊢ new_element →_h h'
          ⟹ h' ⊢ get_shadow_root new_element_ptr →_r None"


interpretation i_new_element_get_shadow_root?:
  l_new_element_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  ⟨proof⟩
declare l_new_element_get_shadow_root_Shadow_DOM_axioms [instances]

lemma new_element_get_shadow_root_is_l_new_element_get_shadow_root [instances]:
  "l_new_element_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  ⟨proof⟩
```

new_shadow_root

```
locale l_new_shadow_root_get_shadow_root_Shadow_DOM =
  l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for type_wf :: "(_) heap ⇒ bool"
    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 get_shadow_root_new_shadow_root:
  "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_h h'
    ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"
  ⟨proof⟩
end

locale l_new_shadow_root_get_shadow_root = l_get_shadow_root +
  assumes get_shadow_root_new_shadow_root:
    "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr
          ⟹ h ⊢ new_ShadowRoot_M →_h h' ⟹ r ∈ get_shadow_root_locs ptr' ⟹ r h h'"

interpretation i_new_shadow_root_get_shadow_root?:
  l_new_shadow_root_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  ⟨proof⟩
declare l_new_shadow_root_get_shadow_root_Shadow_DOM_axioms [instances]

lemma new_shadow_root_get_shadow_root_is_l_new_shadow_root_get_shadow_root [instances]:
  "l_new_shadow_root_get_shadow_root type_wf get_shadow_root get_shadow_root_locs"
  ⟨proof⟩
```

set_shadow_root

```
locale l_set_shadow_root_Shadow_DOM_defs
begin
definition a_set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
  where
    "a_set_shadow_root element_ptr = put_M element_ptr shadow_root_opt_update"

definition a_set_shadow_root_locs :: "(_) element_ptr ⇒ ((_, unit) dom_prog) set"
  where
    "a_set_shadow_root_locs element_ptr ≡ all_args (put_M element_ptr shadow_root_opt_update)"
end

global_interpretation l_set_shadow_root_Shadow_DOM_defs
  defines set_shadow_root = a_set_shadow_root
    and set_shadow_root_locs = a_set_shadow_root_locs
  ⟨proof⟩

locale l_set_shadow_root_defs =
  fixes set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
  fixes set_shadow_root_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set"
```

```
locale l_set_shadow_root_Shadow_DOM =
  l_type_wf type_wf +
  l_set_shadow_root_defs set_shadow_root set_shadow_root_locs +
  l_set_shadow_root_Shadow_DOM_defs
  for type_wf :: "(_) heap ⇒ bool"
    and set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
    and set_shadow_root_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes set_shadow_root_impl: "set_shadow_root = a_set_shadow_root"
  assumes set_shadow_root_locs_impl: "set_shadow_root_locs = a_set_shadow_root_locs"
begin
lemmas set_shadow_root_def = set_shadow_root_impl[unfolded set_shadow_root_def
    a_set_shadow_root_def]
lemmas set_shadow_root_locs_def = set_shadow_root_locs_impl[unfolded set_shadow_root_locs_def
    a_set_shadow_root_locs_def]

lemma set_shadow_root_ok: "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹
  h ⊢ ok (set_shadow_root element_ptr tag)"
  ⟨proof⟩

lemma set_shadow_root_ptr_in_heap:
  "h ⊢ ok (set_shadow_root element_ptr shadow_root) ⟹ element_ptr |∈| element_ptr_kinds h"
  ⟨proof⟩

lemma set_shadow_root_writes:
  "writes (set_shadow_root_locs element_ptr) (set_shadow_root element_ptr tag) h h'"
  ⟨proof⟩

lemma set_shadow_root_pointers_preserved:
  assumes "w ∈ set_shadow_root_locs element_ptr"
  assumes "h ⊢ w →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  ⟨proof⟩

lemma set_shadow_root_types_preserved:
  assumes "w ∈ set_shadow_root_locs element_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  ⟨proof⟩
end

interpretation i_set_shadow_root?: l_set_shadow_root_Shadow_DOM type_wf set_shadow_root
  set_shadow_root_locs
  ⟨proof⟩
declare l_set_shadow_root_Shadow_DOM_axioms [instances]

locale l_set_shadow_root = l_type_wf +l_set_shadow_root_defs +
  assumes set_shadow_root_writes:
    "writes (set_shadow_root_locs element_ptr) (set_shadow_root element_ptr disc_nodes) h h'"
  assumes set_shadow_root_ok:
    "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹
      h ⊢ ok (set_shadow_root element_ptr shadow_root)"
  assumes set_shadow_root_ptr_in_heap:
    "h ⊢ ok (set_shadow_root element_ptr shadow_root) ⟹ element_ptr |∈| element_ptr_kinds h"
  assumes set_shadow_root_pointers_preserved:
    "w ∈ set_shadow_root_locs element_ptr ⟹ h ⊢ w →_h h' ⟹
      object_ptr_kinds h = object_ptr_kinds h'"
  assumes set_shadow_root_types_preserved:
    "w ∈ set_shadow_root_locs element_ptr ⟹ h ⊢ w →_h h' ⟹ type_wf h = type_wf h'"

lemma set_shadow_root_is_l_set_shadow_root [instances]:
  "l_set_shadow_root type_wf set_shadow_root set_shadow_root_locs"
  ⟨proof⟩
```

get_shadow_root **locale** *l_set_shadow_root_get_shadow_root$_{Shadow_DOM}$ =*
l_set_shadow_root$_{Shadow_DOM}$ +
l_get_shadow_root$_{Shadow_DOM}$
begin
lemma *set_shadow_root_get_shadow_root:*
"type_wf h ⟹ h ⊢ set_shadow_root ptr shadow_root_ptr_opt →$_h$ h' ⟹
h' ⊢ get_shadow_root ptr →$_r$ shadow_root_ptr_opt"
⟨*proof*⟩

lemma *set_shadow_root_get_shadow_root_different_pointers: "ptr ≠ ptr' ⟹*
∀w ∈ set_shadow_root_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
⟨*proof*⟩
end

interpretation *i_set_shadow_root_get_shadow_root?: l_set_shadow_root_get_shadow_root$_{Shadow_DOM}$ type_wf*
set_shadow_root set_shadow_root_locs get_shadow_root get_shadow_root_locs
⟨*proof*⟩
declare *l_set_shadow_root_get_shadow_root$_{Shadow_DOM}$_axioms[instances]*

locale *l_set_shadow_root_get_shadow_root =*
l_type_wf +
l_set_shadow_root_defs +
l_get_shadow_root_defs +
assumes *set_shadow_root_get_shadow_root:*
"type_wf h ⟹ h ⊢ set_shadow_root ptr shadow_root_ptr_opt →$_h$ h' ⟹
h' ⊢ get_shadow_root ptr →$_r$ shadow_root_ptr_opt"
assumes *set_shadow_root_get_shadow_root_different_pointers:*
"ptr ≠ ptr' ⟹ w ∈ set_shadow_root_locs ptr ⟹ h ⊢ w →$_h$ h' ⟹
r ∈ get_shadow_root_locs ptr' ⟹ r h h'"

lemma *set_shadow_root_get_shadow_root_is_l_set_shadow_root_get_shadow_root [instances]:*
"l_set_shadow_root_get_shadow_root type_wf set_shadow_root set_shadow_root_locs get_shadow_root
get_shadow_root_locs"
⟨*proof*⟩

set_mode

locale *l_set_mode$_{Shadow_DOM}$_defs*
begin
definition *a_set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"*
where
"a_set_mode shadow_root_ptr = put_M shadow_root_ptr mode_update"

definition *a_set_mode_locs :: "(_) shadow_root_ptr ⇒ ((_, unit) dom_prog) set"*
where
"a_set_mode_locs shadow_root_ptr ≡ all_args (put_M shadow_root_ptr mode_update)"
end

global_interpretation *l_set_mode$_{Shadow_DOM}$_defs*
defines *set_mode = a_set_mode*
and *set_mode_locs = a_set_mode_locs*
⟨*proof*⟩

locale *l_set_mode_defs =*
fixes *set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"*
fixes *set_mode_locs :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog set"*

locale *l_set_mode$_{Shadow_DOM}$ =*
l_type_wf type_wf +
l_set_mode_defs set_mode set_mode_locs +
l_set_mode$_{Shadow_DOM}$_defs
for *type_wf :: "(_) heap ⇒ bool"*

```
    and set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"
    and set_mode_locs :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes set_mode_impl: "set_mode = a_set_mode"
  assumes set_mode_locs_impl: "set_mode_locs = a_set_mode_locs"
begin
lemmas set_mode_def = set_mode_impl[unfolded set_mode_def a_set_mode_def]
lemmas set_mode_locs_def = set_mode_locs_impl[unfolded set_mode_locs_def a_set_mode_locs_def]

lemma set_mode_ok:
  "type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹
    h ⊢ ok (set_mode shadow_root_ptr shadow_root_mode)"
  ⟨proof⟩

lemma set_mode_ptr_in_heap:
  "h ⊢ ok (set_mode shadow_root_ptr shadow_root_mode) ⟹
    shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  ⟨proof⟩

lemma set_mode_writes:
  "writes (set_mode_locs shadow_root_ptr) (set_mode shadow_root_ptr shadow_root_mode) h h'"
  ⟨proof⟩

lemma set_mode_pointers_preserved:
  assumes "w ∈ set_mode_locs element_ptr"
  assumes "h ⊢ w →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  ⟨proof⟩

lemma set_mode_types_preserved:
  assumes "w ∈ set_mode_locs element_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  ⟨proof⟩
end

interpretation i_set_mode?: l_set_mode_Shadow_DOM type_wf set_mode set_mode_locs
  ⟨proof⟩
declare l_set_mode_Shadow_DOM_axioms [instances]

locale l_set_mode = l_type_wf +l_set_mode_defs +
  assumes set_mode_writes:
    "writes (set_mode_locs shadow_root_ptr) (set_mode shadow_root_ptr shadow_root_mode) h h'"
  assumes set_mode_ok:
    "type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹
      h ⊢ ok (set_mode shadow_root_ptr shadow_root_mode)"
  assumes set_mode_ptr_in_heap:
    "h ⊢ ok (set_mode shadow_root_ptr shadow_root_mode) ⟹
      shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  assumes set_mode_pointers_preserved:
    "w ∈ set_mode_locs shadow_root_ptr ⟹
      h ⊢ w →_h h' ⟹ object_ptr_kinds h = object_ptr_kinds h'"
  assumes set_mode_types_preserved:
    "w ∈ set_mode_locs shadow_root_ptr ⟹ h ⊢ w →_h h' ⟹ type_wf h = type_wf h'"

lemma set_mode_is_l_set_mode [instances]: "l_set_mode type_wf set_mode set_mode_locs"
  ⟨proof⟩

get_child_nodes locale l_set_shadow_root_get_child_nodes_Shadow_DOM =
  l_get_child_nodes_Shadow_DOM +
  l_set_shadow_root_Shadow_DOM
begin
lemma set_shadow_root_get_child_nodes:
```

```
  "∀w ∈ set_shadow_root_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
```
⟨*proof*⟩
end

interpretation *i_set_shadow_root_get_child_nodes?: l_set_shadow_root_get_child_nodes*$_{Shadow_DOM}$ *type_wf known_ptr DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs set_shadow_root set_shadow_root_locs*
⟨*proof*⟩
declare *l_set_shadow_root_get_child_nodes*$_{Shadow_DOM}$*_axioms[instances]*

locale *l_set_shadow_root_get_child_nodes = l_set_shadow_root + l_get_child_nodes +*
assumes *set_shadow_root_get_child_nodes:*
```
  "∀w ∈ set_shadow_root_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
```

lemma *set_shadow_root_get_child_nodes_is_l_set_shadow_root_get_child_nodes [instances]:*
```
  "l_set_shadow_root_get_child_nodes type_wf set_shadow_root set_shadow_root_locs known_ptr
    get_child_nodes get_child_nodes_locs"
```
⟨*proof*⟩

get_shadow_root **locale** *l_set_mode_get_shadow_root*$_{Shadow_DOM}$ *=*
l_set_mode$_{Shadow_DOM}$ *+*
l_get_shadow_root$_{Shadow_DOM}$
begin
lemma *set_mode_get_shadow_root:*
```
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
```
⟨*proof*⟩
end

interpretation
i_set_mode_get_shadow_root?: l_set_mode_get_shadow_root$_{Shadow_DOM}$ *type_wf set_mode set_mode_locs get_shadow_root get_shadow_root_locs*
⟨*proof*⟩
declare *l_set_mode_get_shadow_root*$_{Shadow_DOM}$*_axioms[instances]*

locale *l_set_mode_get_shadow_root = l_set_mode + l_get_shadow_root +*
assumes *set_mode_get_shadow_root:*
```
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
```

lemma *set_mode_get_shadow_root_is_l_set_mode_get_shadow_root [instances]:*
```
  "l_set_mode_get_shadow_root type_wf set_mode set_mode_locs get_shadow_root
                                        get_shadow_root_locs"
```
⟨*proof*⟩

get_child_nodes **locale** *l_set_mode_get_child_nodes*$_{Shadow_DOM}$ *=*
l_set_mode$_{Shadow_DOM}$ *+*
l_get_child_nodes$_{Shadow_DOM}$
begin
lemma *set_mode_get_child_nodes:*
```
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
```
⟨*proof*⟩
end

interpretation *i_set_mode_get_child_nodes?: l_set_mode_get_child_nodes*$_{Shadow_DOM}$ *type_wf set_mode set_mode_locs known_ptr DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes get_child_nodes_locs Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs*
⟨*proof*⟩
declare *l_set_mode_get_child_nodes*$_{Shadow_DOM}$*_axioms[instances]*

locale *l_set_mode_get_child_nodes = l_set_mode + l_get_child_nodes +*
assumes *set_mode_get_child_nodes:*
```
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
```

```
lemma set_mode_get_child_nodes_is_l_set_mode_get_child_nodes [instances]:
  "l_set_mode_get_child_nodes type_wf set_mode set_mode_locs known_ptr get_child_nodes
                                        get_child_nodes_locs"
  ⟨proof⟩
```

get_host

```
locale l_get_host_Shadow_DOM_defs =
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs
  for get_shadow_root
    :: "(_::linorder) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_get_host :: "(_) shadow_root_ptr ⇒ (_, (_) element_ptr) dom_prog"  where
  "a_get_host shadow_root_ptr = do {
      host_ptrs ← element_ptr_kinds_M ≫= filter_M (λelement_ptr. do {
        shadow_root_opt ← get_shadow_root element_ptr;
        return (shadow_root_opt = Some shadow_root_ptr)
      });
      (case host_ptrs of host_ptr#[] ⇒ return host_ptr | _ ⇒ error HierarchyRequestError)
    }"
definition "a_get_host_locs ≡ (⋃element_ptr. (get_shadow_root_locs element_ptr)) ∪
  (⋃ptr. {preserved (get_M_Object ptr RObject.nothing)})"

end

global_interpretation l_get_host_Shadow_DOM_defs get_shadow_root get_shadow_root_locs
  defines get_host = "a_get_host"
    and get_host_locs = "a_get_host_locs"
  ⟨proof⟩

locale l_get_host_defs =
  fixes get_host :: "(_) shadow_root_ptr ⇒ (_, (_) element_ptr) dom_prog"
  fixes get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_host_Shadow_DOM =
  l_get_host_Shadow_DOM_defs +
  l_get_host_defs +
  l_get_shadow_root +
  assumes get_host_impl: "get_host = a_get_host"
  assumes get_host_locs_impl: "get_host_locs = a_get_host_locs"
begin
lemmas get_host_def = get_host_impl[unfolded a_get_host_def]
lemmas get_host_locs_def = get_host_locs_impl[unfolded a_get_host_locs_def]

lemma get_host_pure [simp]: "pure (get_host element_ptr) h"
  ⟨proof⟩

lemma get_host_reads: "reads get_host_locs (get_host element_ptr) h h'"
  ⟨proof⟩
end

locale l_get_host = l_get_host_defs +
  assumes get_host_pure [simp]: "pure (get_host element_ptr) h"
  assumes get_host_reads: "reads get_host_locs (get_host node_ptr) h h'"


interpretation i_get_host?: l_get_host_Shadow_DOM get_shadow_root get_shadow_root_locs get_host
  get_host_locs type_wf
  ⟨proof⟩
declare l_get_host_Shadow_DOM_axioms [instances]
```

lemma *get_host_is_l_get_host [instances]: "l_get_host get_host get_host_locs"*
⟨*proof*⟩

get_mode

locale *l_get_mode*$_{Shadow_DOM}$*_defs*
begin
definition *a_get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"*
where
"a_get_mode shadow_root_ptr = get_M shadow_root_ptr mode"

definition *a_get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
where
"a_get_mode_locs shadow_root_ptr ≡ {preserved (get_M shadow_root_ptr mode)}"
end

global_interpretation *l_get_mode*$_{Shadow_DOM}$*_defs*
defines *get_mode = a_get_mode*
and *get_mode_locs = a_get_mode_locs*
⟨*proof*⟩

locale *l_get_mode_defs =*
fixes *get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"*
fixes *get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*

locale *l_get_mode*$_{Shadow_DOM}$ *=*
l_get_mode$_{Shadow_DOM}$*_defs +*
l_get_mode_defs get_mode get_mode_locs +
l_type_wf type_wf
for *get_mode :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, shadow_root_mode) prog"*
and *get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
and *type_wf :: "(_) heap ⇒ bool" +*
assumes *type_wf_impl: "type_wf = ShadowRootClass.type_wf"*
assumes *get_mode_impl: "get_mode = a_get_mode"*
assumes *get_mode_locs_impl: "get_mode_locs = a_get_mode_locs"*
begin
lemmas *get_mode_def = get_mode_impl[unfolded get_mode_def a_get_mode_def]*
lemmas *get_mode_locs_def = get_mode_locs_impl[unfolded get_mode_locs_def a_get_mode_locs_def]*

lemma *get_mode_ok: "type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹*
h ⊢ ok (get_mode shadow_root_ptr)"
⟨*proof*⟩

lemma *get_mode_pure [simp]: "pure (get_mode element_ptr) h"*
⟨*proof*⟩

lemma *get_mode_ptr_in_heap:*
assumes *"h ⊢ get_mode shadow_root_ptr →r children"*
shows *"shadow_root_ptr |∈| shadow_root_ptr_kinds h"*
⟨*proof*⟩

lemma *get_mode_reads: "reads (get_mode_locs element_ptr) (get_mode element_ptr) h h'"*
⟨*proof*⟩
end

interpretation *i_get_mode?: l_get_mode*$_{Shadow_DOM}$ *get_mode get_mode_locs type_wf*
⟨*proof*⟩
declare *l_get_mode*$_{Shadow_DOM}$*_axioms [instances]*

locale *l_get_mode = l_type_wf + l_get_mode_defs +*
assumes *get_mode_reads: "reads (get_mode_locs shadow_root_ptr) (get_mode shadow_root_ptr) h h'"*
assumes *get_mode_ok:*
"type_wf h ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹ h ⊢ ok (get_mode shadow_root_ptr)"

```
  assumes get_mode_ptr_in_heap:
    "h ⊢ ok (get_mode shadow_root_ptr) ⟹ shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  assumes get_mode_pure [simp]: "pure (get_mode shadow_root_ptr) h"

lemma get_mode_is_l_get_mode [instances]: "l_get_mode type_wf get_mode get_mode_locs"
  ⟨proof⟩
```

get_shadow_root_safe

```
locale l_get_shadow_root_safe_Shadow_DOM_defs =
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_get_mode_defs get_mode get_mode_locs
  for get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_get_shadow_root_safe :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
  where
    "a_get_shadow_root_safe element_ptr = do {
      shadow_root_ptr_opt ← get_shadow_root element_ptr;
      (case shadow_root_ptr_opt of
        Some shadow_root_ptr ⇒ do {
          mode ← get_mode shadow_root_ptr;
          (if mode = Open then
            return (Some shadow_root_ptr)
          else
            return None
          )
        } | None ⇒ return None)
    }"

definition a_get_shadow_root_safe_locs
  :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" where
  "a_get_shadow_root_safe_locs element_ptr shadow_root_ptr ≡
    (get_shadow_root_locs element_ptr) ∪ (get_mode_locs shadow_root_ptr)"
end

global_interpretation l_get_shadow_root_safe_Shadow_DOM_defs get_shadow_root get_shadow_root_locs
  get_mode get_mode_locs
  defines get_shadow_root_safe = a_get_shadow_root_safe
    and get_shadow_root_safe_locs = a_get_shadow_root_safe_locs
  ⟨proof⟩

locale l_get_shadow_root_safe_defs =
  fixes get_shadow_root_safe :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
  fixes get_shadow_root_safe_locs ::
    "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_shadow_root_safe_Shadow_DOM =
  l_get_shadow_root_safe_Shadow_DOM_defs get_shadow_root get_shadow_root_locs get_mode get_mode_locs +
  l_get_shadow_root_safe_defs get_shadow_root_safe get_shadow_root_safe_locs +
  l_get_shadow_root  type_wf get_shadow_root get_shadow_root_locs +
  l_get_mode  type_wf get_mode get_mode_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root_safe ::
    "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_safe_locs ::
    "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_shadow_root :: "(_) element_ptr ⇒ (_, (_) shadow_root_ptr option) dom_prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_mode :: "(_) shadow_root_ptr ⇒ (_, shadow_root_mode) dom_prog"
```

```
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
  assumes get_shadow_root_safe_impl: "get_shadow_root_safe = a_get_shadow_root_safe"
  assumes get_shadow_root_safe_locs_impl: "get_shadow_root_safe_locs = a_get_shadow_root_safe_locs"
begin
lemmas get_shadow_root_safe_def =
  get_shadow_root_safe_impl[unfolded get_shadow_root_safe_def a_get_shadow_root_safe_def]
lemmas get_shadow_root_safe_locs_def =
  get_shadow_root_safe_locs_impl[unfolded get_shadow_root_safe_locs_def a_get_shadow_root_safe_locs_def]

lemma get_shadow_root_safe_pure [simp]: "pure (get_shadow_root_safe element_ptr) h"
  ⟨proof⟩
end

interpretation i_get_shadow_root_safe?: l_get_shadow_root_safe_Shadow_DOM type_wf get_shadow_root_safe
  get_shadow_root_safe_locs get_shadow_root get_shadow_root_locs get_mode get_mode_locs
  ⟨proof⟩
declare l_get_shadow_root_safe_Shadow_DOM_axioms [instances]

locale l_get_shadow_root_safe = l_get_shadow_root_safe_defs +
  assumes get_shadow_root_safe_pure [simp]: "pure (get_shadow_root_safe element_ptr) h"

lemma get_shadow_root_safe_is_l_get_shadow_root_safe [instances]:
  "l_get_shadow_root_safe get_shadow_root_safe"
  ⟨proof⟩
```

set_disconnected_nodes

```
locale l_set_disconnected_nodes_Shadow_DOM =
  CD: l_set_disconnected_nodes_Core_DOM type_wf_Core_DOM set_disconnected_nodes set_disconnected_nodes_locs
+
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    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" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin
lemma set_disconnected_nodes_ok:
  "type_wf h ⟹ document_ptr |∈| document_ptr_kinds h ⟹
    h ⊢ ok (set_disconnected_nodes document_ptr node_ptrs)"
  ⟨proof⟩

lemma set_disconnected_nodes_typess_preserved:
  assumes "w ∈ set_disconnected_nodes_locs object_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  ⟨proof⟩
end

interpretation i_set_disconnected_nodes?: l_set_disconnected_nodes_Shadow_DOM type_wf
  DocumentClass.type_wf set_disconnected_nodes set_disconnected_nodes_locs
  ⟨proof⟩
declare l_set_disconnected_nodes_Shadow_DOM_axioms [instances]

lemma set_disconnected_nodes_is_l_set_disconnected_nodes [instances]:
  "l_set_disconnected_nodes type_wf set_disconnected_nodes set_disconnected_nodes_locs"
  ⟨proof⟩
```

get_child_nodes

```
locale l_set_disconnected_nodes_get_child_nodes_Shadow_DOM =
  l_set_disconnected_nodes_Core_DOM type_wf_Core_DOM set_disconnected_nodes set_disconnected_nodes_locs
+
```

```
  l_get_child_nodes_Shadow_DOM type_wf known_ptr type_wf_Core_DOM known_ptr_Core_DOM get_child_nodes
  get_child_nodes_locs get_child_nodes_Core_DOM get_child_nodes_locs_Core_DOM
  for type_wf :: "(_) heap ⇒ bool"
    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 known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool"
    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_child_nodes_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs_Core_DOM :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin

lemma set_disconnected_nodes_get_child_nodes:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_child_nodes_locs ptr'. r h h'))"
  ⟨proof⟩
end

interpretation i_set_disconnected_nodes_get_child_nodes?:
  l_set_disconnected_nodes_get_child_nodes_Shadow_DOM type_wf set_disconnected_nodes
  set_disconnected_nodes_locs known_ptr DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes
  get_child_nodes_locs Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs
  ⟨proof⟩
declare l_set_disconnected_nodes_get_child_nodes_Shadow_DOM_axioms[instances]

lemma set_disconnected_nodes_get_child_nodes_is_l_set_disconnected_nodes_get_child_nodes [instances]:
  "l_set_disconnected_nodes_get_child_nodes set_disconnected_nodes_locs get_child_nodes_locs"
  ⟨proof⟩
```

get_host

```
locale l_set_disconnected_nodes_get_host_Shadow_DOM =
  l_set_disconnected_nodes_Shadow_DOM +
  l_get_shadow_root_Shadow_DOM +
  l_get_host_Shadow_DOM
begin
lemma set_disconnected_nodes_get_host:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_host_locs. r h h'))"
  ⟨proof⟩
end

interpretation i_set_disconnected_nodes_get_host?: l_set_disconnected_nodes_get_host_Shadow_DOM type_wf
  DocumentClass.type_wf set_disconnected_nodes set_disconnected_nodes_locs get_shadow_root
  get_shadow_root_locs get_host get_host_locs
  ⟨proof⟩
declare l_set_disconnected_nodes_get_host_Shadow_DOM_axioms [instances]

locale l_set_disconnected_nodes_get_host = l_set_disconnected_nodes_defs + l_get_host_defs +
  assumes set_disconnected_nodes_get_host:
    "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_host_locs. r h h'))"

lemma set_disconnected_nodes_get_host_is_l_set_disconnected_nodes_get_host [instances]:
  "l_set_disconnected_nodes_get_host set_disconnected_nodes_locs get_host_locs"
  ⟨proof⟩
```

get_tag_name

```
locale l_get_tag_name_Shadow_DOM =
  CD: l_get_tag_name_Core_DOM type_wf_Core_DOM get_tag_name get_tag_name_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ (_, tag_name) dom_prog"
```

```
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin

lemma get_tag_name_ok:
  "type_wf h ⟹ element_ptr |∈| element_ptr_kinds h ⟹ h ⊢ ok (get_tag_name element_ptr)"
  ⟨proof⟩
end

interpretation i_get_tag_name?: l_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name
  get_tag_name_locs
  ⟨proof⟩
declare l_get_tag_name_Shadow_DOM_axioms [instances]

lemma get_tag_name_is_l_get_tag_name [instances]: "l_get_tag_name type_wf get_tag_name
  get_tag_name_locs"
  ⟨proof⟩
```

set_disconnected_nodes locale `l_set_disconnected_nodes_get_tag_name_Shadow_DOM =`
`l_set_disconnected_nodes_Shadow_DOM +`
`l_get_tag_name_Shadow_DOM`

```
begin
lemma set_disconnected_nodes_get_tag_name:
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"
  ⟨proof⟩
end

interpretation i_set_disconnected_nodes_get_tag_name?: l_set_disconnected_nodes_get_tag_name_Shadow_DOM
  type_wf DocumentClass.type_wf set_disconnected_nodes set_disconnected_nodes_locs get_tag_name
  get_tag_name_locs
  ⟨proof⟩
declare l_set_disconnected_nodes_get_tag_name_Shadow_DOM_axioms [instances]

lemma set_disconnected_nodes_get_tag_name_is_l_set_disconnected_nodes_get_tag_name [instances]:
  "l_set_disconnected_nodes_get_tag_name type_wf set_disconnected_nodes set_disconnected_nodes_locs
  get_tag_name get_tag_name_locs"
  ⟨proof⟩
```

set_child_nodes locale `l_set_child_nodes_get_tag_name_Shadow_DOM =`
`l_set_child_nodes_Shadow_DOM +`
`l_get_tag_name_Shadow_DOM`

```
begin
lemma set_child_nodes_get_tag_name:
  "∀w ∈ set_child_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"
  ⟨proof⟩
end

interpretation i_set_child_nodes_get_tag_name?: l_set_child_nodes_get_tag_name_Shadow_DOM type_wf
  known_ptr DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
  Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs get_tag_name
  get_tag_name_locs
  ⟨proof⟩
declare l_set_child_nodes_get_tag_name_Shadow_DOM_axioms [instances]

lemma set_child_nodes_get_tag_name_is_l_set_child_nodes_get_tag_name [instances]:
  "l_set_child_nodes_get_tag_name type_wf set_child_nodes set_child_nodes_locs get_tag_name
  get_tag_name_locs"
  ⟨proof⟩
```

delete_shadow_root locale `l_delete_shadow_root_get_tag_name_Shadow_DOM =`
`l_get_tag_name_Shadow_DOM`

```
begin
```

lemma *get_tag_name_delete_shadow_root: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h'*
⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
⟨*proof*⟩
end

locale *l_delete_shadow_root_get_tag_name = l_get_tag_name_defs +*
assumes *get_tag_name_delete_shadow_root: "h ⊢ delete$_{ShadowRoot}$_M shadow_root_ptr →$_h$ h'*
⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
interpretation *l_delete_shadow_root_get_tag_name$_{Shadow_DOM}$ type_wf DocumentClass.type_wf get_tag_name*
get_tag_name_locs
⟨*proof*⟩

lemma *l_delete_shadow_root_get_tag_name_get_tag_name_locs [instances]: "l_delete_shadow_root_get_tag_name*
get_tag_name_locs"
⟨*proof*⟩

set_shadow_root **locale** *l_set_shadow_root_get_tag_name$_{Shadow_DOM}$ =*
l_set_shadow_root$_{Shadow_DOM}$ +
l_get_tag_name$_{Shadow_DOM}$
begin
lemma *set_shadow_root_get_tag_name:*
"∀w ∈ set_shadow_root_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"
⟨*proof*⟩
end

interpretation *i_set_shadow_root_get_tag_name?: l_set_shadow_root_get_tag_name$_{Shadow_DOM}$ type_wf*
set_shadow_root set_shadow_root_locs DocumentClass.type_wf get_tag_name get_tag_name_locs
⟨*proof*⟩
declare *l_set_shadow_root_get_tag_name$_{Shadow_DOM}$_axioms[instances]*

locale *l_set_shadow_root_get_tag_name = l_set_shadow_root_defs + l_get_tag_name_defs +*
assumes *set_shadow_root_get_tag_name:*
"∀w ∈ set_shadow_root_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"

lemma *set_shadow_root_get_tag_name_is_l_set_shadow_root_get_tag_name [instances]:*
"l_set_shadow_root_get_tag_name set_shadow_root_locs get_tag_name_locs"
⟨*proof*⟩

new_element **locale** *l_new_element_get_tag_name$_{Shadow_DOM}$ =*
l_get_tag_name$_{Shadow_DOM}$ type_wf type_wf$_{Core_DOM}$ get_tag_name get_tag_name_locs
for *type_wf :: "(_) heap ⇒ bool"*
and *type_wf$_{Core_DOM}$ :: "(_) heap ⇒ bool"*
and *get_tag_name :: "(_) element_ptr ⇒ (_, tag_name) dom_prog"*
and *get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
begin
lemma *get_tag_name_new_element:*
"ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →$_r$ new_element_ptr ⟹ h ⊢ new_element →$_h$ h'
⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
⟨*proof*⟩

lemma *new_element_empty_tag_name:*
"h ⊢ new_element →$_r$ new_element_ptr ⟹ h ⊢ new_element →$_h$ h'
⟹ h' ⊢ get_tag_name new_element_ptr →$_r$ ''''"
⟨*proof*⟩
end

locale *l_new_element_get_tag_name = l_new_element + l_get_tag_name +*
assumes *get_tag_name_new_element:*
"ptr' ≠ new_element_ptr ⟹ h ⊢ new_element →$_r$ new_element_ptr
⟹ h ⊢ new_element →$_h$ h' ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
assumes *new_element_empty_tag_name:*
"h ⊢ new_element →$_r$ new_element_ptr ⟹ h ⊢ new_element →$_h$ h'
⟹ h' ⊢ get_tag_name new_element_ptr →$_r$ ''''"

```
interpretation i_new_element_get_tag_name?:
  l_new_element_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name get_tag_name_locs
  ⟨proof⟩
declare l_new_element_get_tag_name_Shadow_DOM_axioms [instances]

lemma new_element_get_tag_name_is_l_new_element_get_tag_name [instances]:
  "l_new_element_get_tag_name type_wf get_tag_name get_tag_name_locs"
  ⟨proof⟩
```

get_shadow_root `locale l_set_mode_get_tag_name_Shadow_DOM =`

```
  l_set_mode_Shadow_DOM +
  l_get_tag_name_Shadow_DOM
begin
lemma set_mode_get_tag_name:
  "∀w ∈ set_mode_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"
  ⟨proof⟩
end

interpretation
  i_set_mode_get_tag_name?: l_set_mode_get_tag_name_Shadow_DOM type_wf
  set_mode set_mode_locs  DocumentClass.type_wf get_tag_name
  get_tag_name_locs
  ⟨proof⟩
declare l_set_mode_get_tag_name_Shadow_DOM_axioms[instances]

locale l_set_mode_get_tag_name = l_set_mode + l_get_tag_name +
  assumes set_mode_get_tag_name:
    "∀w ∈ set_mode_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"

lemma set_mode_get_tag_name_is_l_set_mode_get_tag_name [instances]:
  "l_set_mode_get_tag_name type_wf set_mode set_mode_locs get_tag_name
                                        get_tag_name_locs"
  ⟨proof⟩
```

new_document `locale l_new_document_get_tag_name_Shadow_DOM =`

```
  l_get_tag_name_Shadow_DOM type_wf type_wf_Core_DOM get_tag_name get_tag_name_locs
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, tag_name) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
lemma get_tag_name_new_document:
  "h ⊢ new_document →_r new_document_ptr ⟹ h ⊢ new_document →_h h'
      ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
  ⟨proof⟩
end

locale l_new_document_get_tag_name = l_get_tag_name_defs +
  assumes get_tag_name_new_document:
    "h ⊢ new_document →_r new_document_ptr ⟹ h ⊢ new_document →_h h'
      ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"

interpretation i_new_document_get_tag_name?:
  l_new_document_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name
  get_tag_name_locs
  ⟨proof⟩
declare l_new_document_get_tag_name_Shadow_DOM_def[instances]

lemma new_document_get_tag_name_is_l_new_document_get_tag_name [instances]:
  "l_new_document_get_tag_name get_tag_name_locs"
  ⟨proof⟩
```

new_shadow_root **locale** *l_new_shadow_root_get_tag_name$_{Shadow_DOM}$ =*
l_get_tag_name$_{Shadow_DOM}$
begin
lemma *get_tag_name_new_shadow_root:*
"h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr ⟹ h ⊢ new$_{ShadowRoot}$_M →$_h$ h'
⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
⟨*proof*⟩
end

locale *l_new_shadow_root_get_tag_name = l_get_tag_name +*
assumes *get_tag_name_new_shadow_root:*
"h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr
⟹ h ⊢ new$_{ShadowRoot}$_M →$_h$ h' ⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"

interpretation *i_new_shadow_root_get_tag_name?:*
l_new_shadow_root_get_tag_name$_{Shadow_DOM}$ type_wf DocumentClass.type_wf get_tag_name get_tag_name_locs
⟨*proof*⟩
declare *l_new_shadow_root_get_tag_name$_{Shadow_DOM}$_axioms [instances]*

lemma *new_shadow_root_get_tag_name_is_l_new_shadow_root_get_tag_name [instances]:*
"l_new_shadow_root_get_tag_name type_wf get_tag_name get_tag_name_locs"
⟨*proof*⟩

new_character_data **locale** *l_new_character_data_get_tag_name$_{Shadow_DOM}$ =*
l_get_tag_name$_{Shadow_DOM}$ type_wf type_wf$_{Core_DOM}$ get_tag_name get_tag_name_locs
for *type_wf :: "(_) heap ⇒ bool"*
and *type_wf$_{Core_DOM}$ :: "(_) heap ⇒ bool"*
and *get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, tag_name) prog"*
and *get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
begin
lemma *get_tag_name_new_character_data:*
"h ⊢ new_character_data →$_r$ new_character_data_ptr ⟹ h ⊢ new_character_data →$_h$ h'
⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"
⟨*proof*⟩
end

locale *l_new_character_data_get_tag_name = l_get_tag_name_defs +*
assumes *get_tag_name_new_character_data:*
"h ⊢ new_character_data →$_r$ new_character_data_ptr ⟹ h ⊢ new_character_data →$_h$ h'
⟹ r ∈ get_tag_name_locs ptr' ⟹ r h h'"

interpretation *i_new_character_data_get_tag_name?:*
l_new_character_data_get_tag_name$_{Shadow_DOM}$ type_wf DocumentClass.type_wf get_tag_name
get_tag_name_locs
⟨*proof*⟩
declare *l_new_character_data_get_tag_name$_{Shadow_DOM}$_def[instances]*

lemma *new_character_data_get_tag_name_is_l_new_character_data_get_tag_name [instances]:*
"l_new_character_data_get_tag_name get_tag_name_locs"
⟨*proof*⟩

get_tag_type **locale** *l_set_tag_name_get_tag_name$_{Shadow_DOM}$ = l_get_tag_name$_{Shadow_DOM}$*
+ l_set_tag_name$_{Shadow_DOM}$
begin
lemma *set_tag_name_get_tag_name:*
assumes *"h ⊢ CD.a_set_tag_name element_ptr tag →$_h$ h'"*
shows *"h' ⊢ CD.a_get_tag_name element_ptr →$_r$ tag"*
⟨*proof*⟩

lemma *set_tag_name_get_tag_name_different_pointers:*
assumes *"ptr ≠ ptr'"*
assumes *"w ∈ CD.a_set_tag_name_locs ptr"*
assumes *"h ⊢ w →$_h$ h'"*

```
  assumes "r ∈ CD.a_get_tag_name_locs ptr'"
  shows "r h h'"
  ⟨proof⟩
end

interpretation i_set_tag_name_get_tag_name?:
  l_set_tag_name_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf get_tag_name
  get_tag_name_locs set_tag_name set_tag_name_locs
  ⟨proof⟩
declare l_set_tag_name_get_tag_name_Shadow_DOM_axioms[instances]

lemma set_tag_name_get_tag_name_is_l_set_tag_name_get_tag_name [instances]:
  "l_set_tag_name_get_tag_name  type_wf get_tag_name get_tag_name_locs
                                                    set_tag_name set_tag_name_locs"
  ⟨proof⟩
```

attach_shadow_root

```
locale l_attach_shadow_root_Shadow_DOM_defs =
  l_set_shadow_root_defs set_shadow_root set_shadow_root_locs +
  l_set_mode_defs set_mode set_mode_locs +
  l_get_tag_name_defs get_tag_name get_tag_name_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs
  for 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 get_tag_name :: "(_) element_ptr ⇒ (_, char list) dom_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
definition a_attach_shadow_root ::
  "(_) element_ptr ⇒ shadow_root_mode ⇒ (_, (_) shadow_root_ptr) dom_prog"
  where
    "a_attach_shadow_root element_ptr shadow_root_mode = do {
      tag ← get_tag_name element_ptr;
      (if tag ∉ safe_shadow_root_element_types then error NotSupportedError else return ());
      prev_shadow_root ← get_shadow_root element_ptr;
      (if prev_shadow_root ≠ None then error NotSupportedError else return ());
      new_shadow_root_ptr ← new_ShadowRoot_M;
      set_mode new_shadow_root_ptr shadow_root_mode;
      set_shadow_root element_ptr (Some new_shadow_root_ptr);
      return new_shadow_root_ptr
    }"
end

locale l_attach_shadow_root_defs =
  fixes attach_shadow_root ::
    "(_) element_ptr ⇒ shadow_root_mode ⇒ (_, (_) shadow_root_ptr) dom_prog"

global_interpretation l_attach_shadow_root_Shadow_DOM_defs set_shadow_root set_shadow_root_locs
  set_mode set_mode_locs get_tag_name get_tag_name_locs get_shadow_root get_shadow_root_locs
  defines attach_shadow_root = a_attach_shadow_root
  ⟨proof⟩

locale l_attach_shadow_root_Shadow_DOM =
  l_attach_shadow_root_Shadow_DOM_defs set_shadow_root set_shadow_root_locs set_mode set_mode_locs
  get_tag_name get_tag_name_locs get_shadow_root get_shadow_root_locs +
  l_attach_shadow_root_defs attach_shadow_root +
  l_set_shadow_root_Shadow_DOM type_wf set_shadow_root set_shadow_root_locs +
  l_set_mode type_wf set_mode set_mode_locs +
```

```
  l_get_tag_name type_wf get_tag_name get_tag_name_locs +
  l_get_shadow_root type_wf get_shadow_root get_shadow_root_locs +
  l_known_ptr known_ptr
  for known_ptr :: "(_) object_ptr ⇒ bool"
    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 type_wf :: "(_) heap ⇒ bool"
    and get_tag_name :: "(_) element_ptr ⇒ (_, char list) dom_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" +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
  assumes attach_shadow_root_impl: "attach_shadow_root = a_attach_shadow_root"
begin
lemmas attach_shadow_root_def = a_attach_shadow_root_def[folded attach_shadow_root_impl]

lemma attach_shadow_root_element_ptr_in_heap:
  assumes "h ⊢ ok (attach_shadow_root element_ptr shadow_root_mode)"
  shows "element_ptr |∈| element_ptr_kinds h"
⟨proof⟩

lemma create_shadow_root_known_ptr:
  assumes "h ⊢ attach_shadow_root element_ptr shadow_root_mode →_r new_shadow_root_ptr"
  shows "known_ptr (cast_shadow_root_ptr2object_ptr new_shadow_root_ptr)"
  ⟨proof⟩
end

locale l_attach_shadow_root = l_attach_shadow_root_defs

interpretation
  i_attach_shadow_root?: 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
  ⟨proof⟩
declare l_attach_shadow_root_Shadow_DOM_axioms [instances]
```

get_parent

```
global_interpretation l_get_parent_Core_DOM_defs get_child_nodes get_child_nodes_locs
  defines get_parent = "l_get_parent_Core_DOM_defs.a_get_parent get_child_nodes"
    and get_parent_locs = "l_get_parent_Core_DOM_defs.a_get_parent_locs get_child_nodes_locs"
  ⟨proof⟩

interpretation i_get_parent?: l_get_parent_Core_DOM known_ptr type_wf get_child_nodes
  get_child_nodes_locs known_ptrs get_parent get_parent_locs
  ⟨proof⟩
declare l_get_parent_Core_DOM_axioms [instances]

lemma get_parent_is_l_get_parent [instances]: "l_get_parent type_wf known_ptr known_ptrs get_parent
    get_parent_locs get_child_nodes get_child_nodes_locs"
  ⟨proof⟩
```

set_disconnected_nodes

```
locale l_set_disconnected_nodes_get_parent_Shadow_DOM =
  l_set_disconnected_nodes_get_child_nodes
  + l_set_disconnected_nodes_Shadow_DOM
  + l_get_parent_Core_DOM
begin
lemma set_disconnected_nodes_get_parent [simp]:
```

```
  "∀w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_parent_locs. r h h'))"
  ⟨proof⟩
end
interpretation i_set_disconnected_nodes_get_parent?: l_set_disconnected_nodes_get_parent_Shadow_DOM
  set_disconnected_nodes set_disconnected_nodes_locs get_child_nodes get_child_nodes_locs type_wf
  DocumentClass.type_wf known_ptr known_ptrs get_parent get_parent_locs
  ⟨proof⟩
declare l_set_disconnected_nodes_get_parent_Core_DOM_axioms[instances]

lemma set_disconnected_nodes_get_parent_is_l_set_disconnected_nodes_get_parent [instances]:
  "l_set_disconnected_nodes_get_parent set_disconnected_nodes_locs get_parent_locs"
  ⟨proof⟩
```

get_root_node

```
global_interpretation l_get_root_node_Core_DOM_defs get_parent get_parent_locs
  defines get_root_node = "l_get_root_node_Core_DOM_defs.a_get_root_node get_parent"
    and get_root_node_locs = "l_get_root_node_Core_DOM_defs.a_get_root_node_locs get_parent_locs"
    and get_ancestors = "l_get_root_node_Core_DOM_defs.a_get_ancestors get_parent"
    and get_ancestors_locs = "l_get_root_node_Core_DOM_defs.a_get_ancestors_locs get_parent_locs"
  ⟨proof⟩
declare a_get_ancestors.simps [code]

interpretation i_get_root_node?: l_get_root_node_Core_DOM type_wf known_ptr known_ptrs get_parent
  get_parent_locs get_child_nodes get_child_nodes_locs get_ancestors get_ancestors_locs
  get_root_node get_root_node_locs
  ⟨proof⟩
declare l_get_root_node_Core_DOM_axioms [instances]

lemma get_ancestors_is_l_get_ancestors [instances]: "l_get_ancestors get_ancestors"
  ⟨proof⟩

lemma get_root_node_is_l_get_root_node [instances]: "l_get_root_node get_root_node get_parent"
  ⟨proof⟩
```

get_root_node_si

```
locale l_get_root_node_si_Shadow_DOM_defs =
  l_get_parent_defs get_parent get_parent_locs +
  l_get_host_defs get_host get_host_locs
  for get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_::linorder) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
begin
partial_function (dom_prog) a_get_ancestors_si ::
  "(_::linorder) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  where
    "a_get_ancestors_si ptr = do {
      check_in_heap ptr;
      ancestors ← (case cast_object_ptr2node_ptr ptr of
        Some node_ptr ⇒ do {
          parent_ptr_opt ← get_parent node_ptr;
          (case parent_ptr_opt of
            Some parent_ptr ⇒ a_get_ancestors_si parent_ptr
          | None ⇒ return [])
        }
      | None ⇒ (case cast ptr of
          Some shadow_root_ptr ⇒ do {
            host ← get_host shadow_root_ptr;
            a_get_ancestors_si (cast host)
          } |
          None ⇒ return []));
```

```
    return (ptr # ancestors)
  }"

definition "a_get_ancestors_si_locs = get_parent_locs ∪ get_host_locs"

definition a_get_root_node_si :: "(_) object_ptr ⇒ (_, (_) object_ptr) dom_prog"
  where
    "a_get_root_node_si ptr = do {
      ancestors ← a_get_ancestors_si ptr;
      return (last ancestors)
    }"
definition "a_get_root_node_si_locs = a_get_ancestors_si_locs"
end

locale l_get_ancestors_si_defs =
  fixes get_ancestors_si :: "(_::linorder) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"
  fixes get_ancestors_si_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_root_node_si_defs =
  fixes get_root_node_si :: "(_) object_ptr ⇒ (_, (_) object_ptr) dom_prog"
  fixes get_root_node_si_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_root_node_si_Shadow_DOM =
  l_get_parent +
  l_get_host +
  l_get_root_node_si_Shadow_DOM_defs +
  l_get_ancestors_si_defs +
  l_get_root_node_si_defs +
  assumes get_ancestors_si_impl: "get_ancestors_si = a_get_ancestors_si"
  assumes get_ancestors_si_locs_impl: "get_ancestors_si_locs = a_get_ancestors_si_locs"
  assumes get_root_node_si_impl: "get_root_node_si = a_get_root_node_si"
  assumes get_root_node_si_locs_impl: "get_root_node_si_locs = a_get_root_node_si_locs"
begin
lemmas get_ancestors_si_def = a_get_ancestors_si.simps[folded get_ancestors_si_impl]
lemmas get_ancestors_si_locs_def = a_get_ancestors_si_locs_def[folded get_ancestors_si_locs_impl]
lemmas get_root_node_si_def =
  a_get_root_node_si_def[folded get_root_node_si_impl get_ancestors_si_impl]
lemmas get_root_node_si_locs_def =
  a_get_root_node_si_locs_def[folded get_root_node_si_locs_impl get_ancestors_si_locs_impl]

lemma get_ancestors_si_pure [simp]:
  "pure (get_ancestors_si ptr) h"
⟨proof⟩


lemma get_root_node_si_pure [simp]: "pure (get_root_node_si ptr) h"
  ⟨proof⟩


lemma get_ancestors_si_ptr_in_heap:
  assumes "h ⊢ ok (get_ancestors_si ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  ⟨proof⟩

lemma get_ancestors_si_ptr:
  assumes "h ⊢ get_ancestors_si ptr →r ancestors"
  shows "ptr ∈ set ancestors"
  ⟨proof⟩


lemma get_ancestors_si_never_empty:
  assumes "h ⊢ get_ancestors_si child →r ancestors"
  shows "ancestors ≠ []"
```

⟨*proof*⟩

lemma *get_root_node_si_no_parent:*
"h ⊢ get_parent node_ptr →r None ⟹ h ⊢ get_root_node_si (cast node_ptr) →r cast node_ptr"
⟨*proof*⟩

lemma *get_root_node_si_root_not_shadow_root:*
assumes *"h ⊢ get_root_node_si ptr →r root"*
shows *"¬ is_shadow_root_ptr$_{object_ptr}$ root"*
⟨*proof*⟩
end

global_interpretation *l_get_root_node_si$_{Shadow_DOM}$_defs get_parent get_parent_locs get_host get_host_locs*
defines *get_root_node_si = a_get_root_node_si*
and *get_root_node_si_locs = a_get_root_node_si_locs*
and *get_ancestors_si = a_get_ancestors_si*
and *get_ancestors_si_locs = a_get_ancestors_si_locs*
⟨*proof*⟩
declare *a_get_ancestors_si.simps [code]*

interpretation *i_get_root_node_si?: l_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
⟨*proof*⟩
declare *l_get_root_node_si$_{Shadow_DOM}$_axioms[instances]*

lemma *get_ancestors_si_is_l_get_ancestors [instances]: "l_get_ancestors get_ancestors_si"*
⟨*proof*⟩

lemma *get_root_node_si_is_l_get_root_node [instances]: "l_get_root_node get_root_node_si get_parent"*
⟨*proof*⟩

set_disconnected_nodes **locale** *l_set_disconnected_nodes_get_ancestors_si$_{Core_DOM}$ =*
l_set_disconnected_nodes_get_parent
+ l_get_root_node_si$_{Shadow_DOM}$
+ l_set_disconnected_nodes$_{Shadow_DOM}$
+ l_set_disconnected_nodes_get_host
begin
lemma *set_disconnected_nodes_get_ancestors_si:*
"∀ w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_ancestors_si_locs. r h h'))"
⟨*proof*⟩
end

locale *l_set_disconnected_nodes_get_ancestors_si =*
l_set_disconnected_nodes_defs +
l_get_ancestors_si_defs +
assumes *set_disconnected_nodes_get_ancestors_si:*
"∀ w ∈ set_disconnected_nodes_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_ancestors_si_locs. r h h'))"

interpretation *i_set_disconnected_nodes_get_ancestors_si?:*
l_set_disconnected_nodes_get_ancestors_si$_{Core_DOM}$ set_disconnected_nodes set_disconnected_nodes_locs
get_parent get_parent_locs type_wf known_ptr known_ptrs 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 DocumentClass.type_wf

⟨*proof*⟩
declare *l_set_disconnected_nodes_get_ancestors_si$_{Core_DOM}$_axioms[instances]*

lemma *set_disconnected_nodes_get_ancestors_si_is_l_set_disconnected_nodes_get_ancestors_si [instances]:*
"l_set_disconnected_nodes_get_ancestors_si set_disconnected_nodes_locs get_ancestors_si_locs"
⟨*proof*⟩

get_attribute

lemma *get_attribute_is_l_get_attribute [instances]:*
"l_get_attribute type_wf get_attribute get_attribute_locs"
⟨*proof*⟩

to_tree_order

global_interpretation *l_to_tree_order*$_{Core_DOM}$*_defs get_child_nodes get_child_nodes_locs* **defines**
to_tree_order = "l_to_tree_order$_{Core_DOM}$*_defs.a_to_tree_order get_child_nodes"* ⟨*proof*⟩
declare *a_to_tree_order.simps [code]*

interpretation *i_to_tree_order?: l_to_tree_order*$_{Core_DOM}$ *ShadowRootClass.known_ptr*
ShadowRootClass.type_wf Shadow_DOM.get_child_nodes Shadow_DOM.get_child_nodes_locs to_tree_order
⟨*proof*⟩
declare *l_to_tree_order*$_{Core_DOM}$*_axioms [instances]*

to_tree_order_si

locale *l_to_tree_order_si*$_{Core_DOM}$*_defs =*
l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
l_get_shadow_root_defs get_shadow_root get_shadow_root_locs
for *get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"*
and *get_child_nodes_locs :: "(_) object_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
partial_function *(dom_prog) a_to_tree_order_si :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"*
where
"a_to_tree_order_si ptr = (do {
children ← get_child_nodes ptr;
shadow_root_part ← (case cast ptr of
Some element_ptr ⇒ do {
shadow_root_opt ← get_shadow_root element_ptr;
(case shadow_root_opt of
Some shadow_root_ptr ⇒ return [cast shadow_root_ptr]
| None ⇒ return [])
} |
None ⇒ return []);
treeorders ← map_M a_to_tree_order_si ((map (cast) children) @ shadow_root_part);
return (ptr # concat treeorders)
})"
end

locale *l_to_tree_order_si_defs =*
fixes *to_tree_order_si :: "(_) object_ptr ⇒ (_, (_) object_ptr list) dom_prog"*

global_interpretation *l_to_tree_order_si*$_{Core_DOM}$*_defs get_child_nodes get_child_nodes_locs*
get_shadow_root get_shadow_root_locs
defines *to_tree_order_si = "a_to_tree_order_si"* ⟨*proof*⟩
declare *a_to_tree_order_si.simps [code]*

locale *l_to_tree_order_si*$_{Shadow_DOM}$ *=*
l_to_tree_order_si_defs +
l_to_tree_order_si$_{Core_DOM}$*_defs +*
l_get_child_nodes +
l_get_shadow_root +
assumes *to_tree_order_si_impl: "to_tree_order_si = a_to_tree_order_si"*
begin
lemmas *to_tree_order_si_def = a_to_tree_order_si.simps[folded to_tree_order_si_impl]*

lemma *to_tree_order_si_pure [simp]: "pure (to_tree_order_si ptr) h"*
⟨*proof*⟩

end

interpretation *i_to_tree_order_si?: l_to_tree_order_si*$_{Shadow_DOM}$ *to_tree_order_si get_child_nodes get_child_nodes_locs get_shadow_root get_shadow_root_locs type_wf known_ptr*
⟨*proof*⟩
declare *l_to_tree_order_si*$_{Shadow_DOM}$*_axioms [instances]*

first_in_tree_order

global_interpretation *l_first_in_tree_order*$_{Core_DOM}$*_defs to_tree_order* **defines**
first_in_tree_order = "l_first_in_tree_order$_{Core_DOM}$*_defs.a_first_in_tree_order to_tree_order"* ⟨*proof*⟩

interpretation *i_first_in_tree_order?: l_first_in_tree_order*$_{Core_DOM}$ *to_tree_order first_in_tree_order*
⟨*proof*⟩
declare *l_first_in_tree_order*$_{Core_DOM}$*_axioms [instances]*

lemma *to_tree_order_is_l_to_tree_order [instances]: "l_to_tree_order to_tree_order"*
⟨*proof*⟩

first_in_tree_order

global_interpretation *l_dummy* **defines**
first_in_tree_order_si = "l_first_in_tree_order$_{Core_DOM}$*_defs.a_first_in_tree_order to_tree_order_si"*
⟨*proof*⟩

get_element_by

global_interpretation *l_get_element_by*$_{Core_DOM}$*_defs to_tree_order first_in_tree_order get_attribute get_attribute_locs*
defines
get_element_by_id =
"l_get_element_by$_{Core_DOM}$*_defs.a_get_element_by_id first_in_tree_order get_attribute"* **and**
get_elements_by_class_name =
"l_get_element_by$_{Core_DOM}$*_defs.a_get_elements_by_class_name to_tree_order get_attribute"* **and**
get_elements_by_tag_name =
"l_get_element_by$_{Core_DOM}$*_defs.a_get_elements_by_tag_name to_tree_order"* ⟨*proof*⟩

interpretation *i_get_element_by?: l_get_element_by*$_{Core_DOM}$ *to_tree_order first_in_tree_order get_attribute get_attribute_locs get_element_by_id get_elements_by_class_name get_elements_by_tag_name type_wf*
⟨*proof*⟩
declare *l_get_element_by*$_{Core_DOM}$*_axioms[instances]*

lemma *get_element_by_is_l_get_element_by [instances]:*
"l_get_element_by get_element_by_id get_elements_by_tag_name to_tree_order"
⟨*proof*⟩

get_element_by_si

global_interpretation *l_dummy* **defines**
get_element_by_id_si =
"l_get_element_by$_{Core_DOM}$*_defs.a_get_element_by_id first_in_tree_order_si get_attribute"* **and**
get_elements_by_class_name_si =
"l_get_element_by$_{Core_DOM}$*_defs.a_get_elements_by_class_name to_tree_order_si get_attribute"* **and**
get_elements_by_tag_name_si =
"l_get_element_by$_{Core_DOM}$*_defs.a_get_elements_by_tag_name to_tree_order_si"*
⟨*proof*⟩

find_slot

locale *l_find_slot*$_{Shadow_DOM}$*_defs =*
l_get_parent_defs get_parent get_parent_locs +
l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +

```
  l_get_mode_defs get_mode get_mode_locs +
  l_get_attribute_defs get_attribute get_attribute_locs +
  l_get_tag_name_defs get_tag_name get_tag_name_locs +
  l_first_in_tree_order_defs first_in_tree_order
  for get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_::linorder) object_ptr option) prog"
    and get_parent_locs :: "((_) 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"
    and get_mode :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, shadow_root_mode) prog"
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_attribute :: "(_) element_ptr ⇒ char list ⇒ ((_) heap, exception, char list option) prog"
    and get_attribute_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and first_in_tree_order ::
    "(_) object_ptr ⇒ ((_) object_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog) ⇒
    ((_) heap, exception, (_) element_ptr option) prog"
begin
definition a_find_slot :: "bool ⇒ (_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"
  where
    "a_find_slot open_flag slotable = do {
      parent_opt ← get_parent slotable;
      (case parent_opt of
        Some parent ⇒
          if is_element_ptr_kind parent
          then do {
            shadow_root_ptr_opt ← get_shadow_root (the (cast parent));
            (case shadow_root_ptr_opt of
              Some shadow_root_ptr ⇒ do {
                shadow_root_mode ← get_mode shadow_root_ptr;
                if open_flag ∧ shadow_root_mode ≠ Open
                then return None
                else first_in_tree_order (cast shadow_root_ptr) (λptr. if is_element_ptr_kind ptr
                  then do {
                    tag ← get_tag_name (the (cast ptr));
                    name_attr ← get_attribute (the (cast ptr)) ''name'';
                    slotable_name_attr ← (if is_element_ptr_kind slotable
                                          then get_attribute (the (cast slotable)) ''slot''
                                          else return None);
                    (if (tag = ''slot'' ∧ (name_attr = slotable_name_attr ∨
                        (name_attr = None ∧ slotable_name_attr = Some '''') ∨
                        (name_attr = Some '''' ∧ slotable_name_attr = None)))
                    then return (Some (the (cast ptr)))
                    else return None)}
                  else return None)}
              | None ⇒ return None)}
          else return None
      | _ ⇒ return None)}"

definition a_assigned_slot :: "(_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"
  where
    "a_assigned_slot = a_find_slot True"
end

global_interpretation l_find_slot_Shadow_DOM_defs get_parent get_parent_locs get_shadow_root
  get_shadow_root_locs get_mode get_mode_locs get_attribute get_attribute_locs get_tag_name
  get_tag_name_locs first_in_tree_order
  defines find_slot = a_find_slot
    and assigned_slot = a_assigned_slot
  ⟨proof⟩

locale l_find_slot_defs =
  fixes find_slot :: "bool ⇒ (_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"
```

```
    and assigned_slot :: "(_) node_ptr ⇒ (_, (_) element_ptr option) dom_prog"

locale l_find_slot_Shadow_DOM =
  l_find_slot_Shadow_DOM_defs +
  l_find_slot_defs +
  l_get_parent +
  l_get_shadow_root +
  l_get_mode +
  l_get_attribute +
  l_get_tag_name +
  l_to_tree_order +
  l_first_in_tree_order_Core_DOM +
  assumes find_slot_impl: "find_slot = a_find_slot"
  assumes assigned_slot_impl: "assigned_slot = a_assigned_slot"
begin
lemmas find_slot_def = find_slot_impl[unfolded a_find_slot_def]
lemmas assigned_slot_def = assigned_slot_impl[unfolded a_assigned_slot_def]

lemma find_slot_ptr_in_heap:
  assumes "h ⊢ find_slot open_flag slotable →_r slot_opt"
  shows "slotable |∈| node_ptr_kinds h"
  ⟨proof⟩

lemma find_slot_slot_in_heap:
  assumes "h ⊢ find_slot open_flag slotable →_r Some slot"
  shows "slot |∈| element_ptr_kinds h"
  ⟨proof⟩

lemma find_slot_pure [simp]: "pure (find_slot open_flag slotable) h"
  ⟨proof⟩
end

interpretation i_find_slot?: l_find_slot_Shadow_DOM get_parent get_parent_locs get_shadow_root
  get_shadow_root_locs get_mode get_mode_locs get_attribute get_attribute_locs get_tag_name
  get_tag_name_locs first_in_tree_order find_slot assigned_slot type_wf known_ptr known_ptrs
  get_child_nodes get_child_nodes_locs to_tree_order
  ⟨proof⟩
declare l_find_slot_Shadow_DOM_axioms [instances]

locale l_find_slot = l_find_slot_defs +
  assumes find_slot_ptr_in_heap:
    "h ⊢ find_slot open_flag slotable →_r slot_opt ⟹ slotable |∈| node_ptr_kinds h"
  assumes find_slot_slot_in_heap:
    "h ⊢ find_slot open_flag slotable →_r Some slot ⟹ slot |∈| element_ptr_kinds h"
  assumes find_slot_pure [simp]:
    "pure (find_slot open_flag slotable) h"

lemma find_slot_is_l_find_slot [instances]: "l_find_slot find_slot"
  ⟨proof⟩
```

get_disconnected_nodes

```
locale l_get_disconnected_nodes_Shadow_DOM =
  CD: l_get_disconnected_nodes_Core_DOM type_wf_Core_DOM get_disconnected_nodes
  get_disconnected_nodes_locs +
  l_type_wf type_wf
  for type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ (_, (_) node_ptr list) dom_prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes type_wf_impl: "type_wf = ShadowRootClass.type_wf"
begin
```

lemma *get_disconnected_nodes_ok:*
"type_wf h ⟹ document_ptr |∈| document_ptr_kinds h ⟹
h ⊢ ok (get_disconnected_nodes document_ptr)"
⟨*proof*⟩
end

interpretation *i_get_disconnected_nodes?: l_get_disconnected_nodes*$_{Shadow_DOM}$ *type_wf*
DocumentClass.type_wf get_disconnected_nodes get_disconnected_nodes_locs
⟨*proof*⟩
declare *l_get_disconnected_nodes*$_{Shadow_DOM}$*_axioms [instances]*

lemma *get_disconnected_nodes_is_l_get_disconnected_nodes [instances]:*
"l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs"
⟨*proof*⟩

set_child_nodes **locale** *l_set_child_nodes_get_disconnected_nodes*$_{Shadow_DOM}$ *=*
l_set_child_nodes$_{Shadow_DOM}$ *+*
l_get_disconnected_nodes$_{Shadow_DOM}$
begin
lemma *set_child_nodes_get_disconnected_nodes:*
"∀w ∈ set_child_nodes_locs ptr. (h ⊢ w →$_h$ *h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"*
⟨*proof*⟩
end

interpretation
i_set_child_nodes_get_disconnected_nodes?: l_set_child_nodes_get_disconnected_nodes$_{Shadow_DOM}$ *type_wf*
known_ptr DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs
get_disconnected_nodes get_disconnected_nodes_locs
⟨*proof*⟩
declare *l_set_child_nodes_get_disconnected_nodes*$_{Shadow_DOM}$*_axioms[instances]*

lemma *set_child_nodes_get_disconnected_nodes_is_l_set_child_nodes_get_disconnected_nodes [instances]:*
"l_set_child_nodes_get_disconnected_nodes type_wf set_child_nodes set_child_nodes_locs
get_disconnected_nodes get_disconnected_nodes_locs"
⟨*proof*⟩

set_disconnected_nodes **lemma** *set_disconnected_nodes_get_disconnected_nodes_l_set_disconnected_nodes_get_discon*
[instances]:
"l_set_disconnected_nodes_get_disconnected_nodes ShadowRootClass.type_wf get_disconnected_nodes
get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs"
⟨*proof*⟩

delete_shadow_root **locale** *l_delete_shadow_root_get_disconnected_nodes*$_{Shadow_DOM}$ *=*
l_get_disconnected_nodes$_{Shadow_DOM}$
begin
lemma *get_disconnected_nodes_delete_shadow_root: "h ⊢ delete*$_{ShadowRoot}$*_M shadow_root_ptr →*$_h$ *h'*
⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"
⟨*proof*⟩
end

locale *l_delete_shadow_root_get_disconnected_nodes = l_get_disconnected_nodes_defs +*
assumes *get_disconnected_nodes_delete_shadow_root: "h ⊢ delete*$_{ShadowRoot}$*_M shadow_root_ptr →*$_h$ *h'*
⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"
interpretation *l_delete_shadow_root_get_disconnected_nodes*$_{Shadow_DOM}$ *type_wf DocumentClass.type_wf*
get_disconnected_nodes get_disconnected_nodes_locs
⟨*proof*⟩

lemma *l_delete_shadow_root_get_disconnected_nodes_get_disconnected_nodes_locs [instances]: "l_delete_shadow_root_*
get_disconnected_nodes_locs"
⟨*proof*⟩

set_shadow_root **locale** *l_set_shadow_root_get_disconnected_nodes*$_{Shadow_DOM}$ =
l_set_shadow_root$_{Shadow_DOM}$ +
l_get_disconnected_nodes$_{Shadow_DOM}$
begin
lemma *set_shadow_root_get_disconnected_nodes:*
"∀ w ∈ set_shadow_root_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_disconnected_nodes_locs ptr'. r h h'))"
⟨*proof*⟩
end

interpretation *i_set_shadow_root_get_disconnected_nodes?:*
l_set_shadow_root_get_disconnected_nodes$_{Shadow_DOM}$ *type_wf set_shadow_root set_shadow_root_locs*
DocumentClass.type_wf get_disconnected_nodes get_disconnected_nodes_locs
⟨*proof*⟩
declare *l_set_shadow_root_get_disconnected_nodes*$_{Shadow_DOM}$*_axioms[instances]*

locale *l_set_shadow_root_get_disconnected_nodes =*
l_set_shadow_root_defs +
l_get_disconnected_nodes_defs +
assumes *set_shadow_root_get_disconnected_nodes:*
"∀ w ∈ set_shadow_root_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_disconnected_nodes_locs ptr'. r h h'))"

lemma *set_shadow_root_get_disconnected_nodes_is_l_set_shadow_root_get_disconnected_nodes [instances]:*
"l_set_shadow_root_get_disconnected_nodes set_shadow_root_locs get_disconnected_nodes_locs"
⟨*proof*⟩

set_mode **locale** *l_set_mode_get_disconnected_nodes*$_{Shadow_DOM}$ =
l_set_mode$_{Shadow_DOM}$ +
l_get_disconnected_nodes$_{Shadow_DOM}$
begin
lemma *set_mode_get_disconnected_nodes:*
"∀ w ∈ set_mode_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_disconnected_nodes_locs ptr'. r h h'))"
⟨*proof*⟩
end

interpretation *i_set_mode_get_disconnected_nodes?: l_set_mode_get_disconnected_nodes*$_{Shadow_DOM}$ *type_wf*
set_mode set_mode_locs DocumentClass.type_wf get_disconnected_nodes get_disconnected_nodes_locs
⟨*proof*⟩
declare *l_set_mode_get_disconnected_nodes*$_{Shadow_DOM}$*_axioms[instances]*

locale *l_set_mode_get_disconnected_nodes = l_set_mode + l_get_disconnected_nodes +*
assumes *set_mode_get_disconnected_nodes:*
"∀ w ∈ set_mode_locs ptr. (h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_disconnected_nodes_locs ptr'. r h h'))"

lemma *set_mode_get_disconnected_nodes_is_l_set_mode_get_disconnected_nodes [instances]:*
"l_set_mode_get_disconnected_nodes type_wf set_mode set_mode_locs get_disconnected_nodes
get_disconnected_nodes_locs"
⟨*proof*⟩

new_shadow_root **locale** *l_new_shadow_root_get_disconnected_nodes*$_{Shadow_DOM}$ =
l_get_disconnected_nodes$_{Shadow_DOM}$ *type_wf type_wf*$_{Core_DOM}$ *get_disconnected_nodes*
get_disconnected_nodes_locs
for *type_wf :: "(_) heap ⇒ bool"*
and *type_wf*$_{Core_DOM}$ *:: "(_) heap ⇒ bool"*
and *get_disconnected_nodes :: "(_) document_ptr ⇒ (_, (_) node_ptr list) dom_prog"*
and *get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
begin
lemma *get_disconnected_nodes_new_shadow_root:*
"h ⊢ new$_{ShadowRoot}$_M →$_r$ new_shadow_root_ptr ⟹ h ⊢ new$_{ShadowRoot}$_M →$_h$ h'
⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"
⟨*proof*⟩
end

interpretation *i_new_shadow_root_get_disconnected_nodes?:*

```
  l_new_shadow_root_get_disconnected_nodes_Shadow_DOM type_wf DocumentClass.type_wf
  get_disconnected_nodes get_disconnected_nodes_locs
  ⟨proof⟩
declare l_new_shadow_root_get_disconnected_nodes_Shadow_DOM_axioms[instances]

locale l_new_shadow_root_get_disconnected_nodes = l_get_disconnected_nodes_defs +
  assumes get_disconnected_nodes_new_shadow_root:
    "h ⊢ new_ShadowRoot_M →_r new_shadow_root_ptr ⟹ h ⊢ new_ShadowRoot_M →_h h'
    ⟹ r ∈ get_disconnected_nodes_locs ptr' ⟹ r h h'"

lemma new_shadow_root_get_disconnected_nodes_is_l_new_shadow_root_get_disconnected_nodes [instances]:
  "l_new_shadow_root_get_disconnected_nodes get_disconnected_nodes_locs"
  ⟨proof⟩
```

remove_shadow_root

```
locale l_remove_shadow_root_Shadow_DOM_defs =
  l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_set_shadow_root_defs set_shadow_root set_shadow_root_locs +
  l_get_disconnected_nodes_defs get_disconnected_nodes get_disconnected_nodes_locs
  for get_child_nodes :: "(_) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_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"
    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 get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin

definition a_remove_shadow_root :: "(_) element_ptr ⇒ (_, unit) dom_prog" where
  "a_remove_shadow_root element_ptr = do {
    shadow_root_ptr_opt ← get_shadow_root element_ptr;
    (case shadow_root_ptr_opt of
      Some shadow_root_ptr ⇒ do {
        children ← get_child_nodes (cast shadow_root_ptr);
        (if children = []
        then do {
          set_shadow_root element_ptr None;
          delete_M shadow_root_ptr
        } else do {
          error HierarchyRequestError
        })
      } |
      None ⇒ error HierarchyRequestError)
    }"

definition a_remove_shadow_root_locs ::
  "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_, unit) dom_prog) set"
  where
    "a_remove_shadow_root_locs element_ptr shadow_root_ptr ≡
      set_shadow_root_locs element_ptr ∪ {delete_M shadow_root_ptr}"
end

global_interpretation l_remove_shadow_root_Shadow_DOM_defs get_child_nodes get_child_nodes_locs
  get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs get_disconnected_nodes
  get_disconnected_nodes_locs
  defines remove_shadow_root = "a_remove_shadow_root"
    and remove_shadow_root_locs = a_remove_shadow_root_locs
  ⟨proof⟩
```

locale *l_remove_shadow_root_defs =*
fixes *remove_shadow_root :: "(_) element_ptr ⇒ (_, unit) dom_prog"*
fixes *remove_shadow_root_locs ::*
"(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_, unit) dom_prog) set"

locale *l_remove_shadow_root*$_{Shadow_DOM}$ *=*
l_remove_shadow_root$_{Shadow_DOM}$*_defs +*
l_remove_shadow_root_defs +
l_get_shadow_root$_{Shadow_DOM}$ *+*
l_set_shadow_root$_{Shadow_DOM}$ *+*
l_get_child_nodes +
l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs +
assumes *remove_shadow_root_impl: "remove_shadow_root = a_remove_shadow_root"*
assumes *remove_shadow_root_locs_impl: "remove_shadow_root_locs = a_remove_shadow_root_locs"*
begin
lemmas *remove_shadow_root_def =*
remove_shadow_root_impl[unfolded remove_shadow_root_def a_remove_shadow_root_def]
lemmas *remove_shadow_root_locs_def =*
remove_shadow_root_locs_impl[unfolded remove_shadow_root_locs_def a_remove_shadow_root_locs_def]

lemma *remove_shadow_root_writes:*
"writes (remove_shadow_root_locs element_ptr (the |h ⊢ get_shadow_root element_ptr|$_r$))
(remove_shadow_root element_ptr) h h'"
⟨*proof*⟩
end

interpretation *i_remove_shadow_root?: l_remove_shadow_root*$_{Shadow_DOM}$ *get_child_nodes*
get_child_nodes_locs get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs
get_disconnected_nodes get_disconnected_nodes_locs remove_shadow_root remove_shadow_root_locs
type_wf known_ptr
⟨*proof*⟩
declare *l_remove_shadow_root*$_{Shadow_DOM}$*_axioms [instances]*

get_child_nodes **locale** *l_remove_shadow_root_get_child_nodes*$_{Shadow_DOM}$ *=*
l_get_child_nodes$_{Shadow_DOM}$ *+*
l_remove_shadow_root$_{Shadow_DOM}$
begin

lemma *remove_shadow_root_get_child_nodes_different_pointers:*
assumes *"ptr ≠ cast shadow_root_ptr"*
assumes *"w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr"*
assumes *"h ⊢ w →$_h$ h'"*
assumes *"r ∈ get_child_nodes_locs ptr"*
shows *"r h h'"*
⟨*proof*⟩
end

locale *l_remove_shadow_root_get_child_nodes = l_get_child_nodes_defs + l_remove_shadow_root_defs +*
assumes *remove_shadow_root_get_child_nodes_different_pointers:*
"ptr ≠ cast shadow_root_ptr ⟹ w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr ⟹
h ⊢ w →$_h$ h' ⟹ r ∈ get_child_nodes_locs ptr ⟹ r h h'"

interpretation *i_remove_shadow_root_get_child_nodes?: l_remove_shadow_root_get_child_nodes*$_{Shadow_DOM}$
type_wf known_ptr DocumentClass.type_wf DocumentClass.known_ptr get_child_nodes
get_child_nodes_locs Core_DOM_Functions.get_child_nodes Core_DOM_Functions.get_child_nodes_locs
get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs get_disconnected_nodes
get_disconnected_nodes_locs remove_shadow_root remove_shadow_root_locs
⟨*proof*⟩
declare *l_remove_shadow_root_get_child_nodes*$_{Shadow_DOM}$*_axioms[instances]*

lemma *remove_shadow_root_get_child_nodes_is_l_remove_shadow_root_get_child_nodes [instances]:*
"l_remove_shadow_root_get_child_nodes get_child_nodes_locs remove_shadow_root_locs"

⟨*proof*⟩

get_tag_name **locale** *l_remove_shadow_root_get_tag_name*$_{Shadow_DOM}$ =
l_get_tag_name$_{Shadow_DOM}$ +
l_remove_shadow_root$_{Shadow_DOM}$
begin

lemma *remove_shadow_root_get_tag_name:*
assumes *"w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr"*
assumes *"h ⊢ w →$_h$ h'"*
assumes *"r ∈ get_tag_name_locs ptr"*
shows *"r h h'"*
⟨*proof*⟩
end

locale *l_remove_shadow_root_get_tag_name = l_get_tag_name_defs + l_remove_shadow_root_defs +*
assumes *remove_shadow_root_get_tag_name:*
"w ∈ remove_shadow_root_locs element_ptr shadow_root_ptr ⟹ h ⊢ w →$_h$ h' ⟹
r ∈ get_tag_name_locs ptr ⟹ r h h'"

interpretation *i_remove_shadow_root_get_tag_name?: l_remove_shadow_root_get_tag_name*$_{Shadow_DOM}$ *type_wf*
DocumentClass.type_wf get_tag_name get_tag_name_locs get_child_nodes get_child_nodes_locs
get_shadow_root get_shadow_root_locs set_shadow_root set_shadow_root_locs get_disconnected_nodes
get_disconnected_nodes_locs remove_shadow_root remove_shadow_root_locs known_ptr
⟨*proof*⟩
declare *l_remove_shadow_root_get_tag_name*$_{Shadow_DOM}$*_axioms[instances]*

lemma *remove_shadow_root_get_tag_name_is_l_remove_shadow_root_get_tag_name [instances]:*
"l_remove_shadow_root_get_tag_name get_tag_name_locs remove_shadow_root_locs"
⟨*proof*⟩

get_owner_document

locale *l_get_owner_document*$_{Shadow_DOM}$*_defs =*
l_get_host_defs get_host get_host_locs +
CD: l_get_owner_document$_{Core_DOM}$*_defs get_root_node get_root_node_locs get_disconnected_nodes*
get_disconnected_nodes_locs
for *get_root_node :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"*
and *get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) 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 *get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"*
and *get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"*
begin
definition *a_get_owner_document*$_{shadow_root_ptr}$ *::*
"(_) shadow_root_ptr ⇒ unit ⇒ (_, (_) document_ptr) dom_prog"
where
"a_get_owner_document$_{shadow_root_ptr}$ *shadow_root_ptr _ = do {*
host ← get_host shadow_root_ptr;
CD.a_get_owner_document$_{node_ptr}$ *(cast host) ()*
}"

definition *a_get_owner_document_tups :: "(((_) object_ptr ⇒ bool) × ((_) object_ptr ⇒ unit*
⇒ (_, (_) document_ptr) dom_prog)) list"
where
"a_get_owner_document_tups = [(is_shadow_root_ptr, a_get_owner_document$_{shadow_root_ptr}$ *∘ the ∘ cast)]"*

definition *a_get_owner_document :: "(_::linorder) object_ptr ⇒ (_, (_) document_ptr) dom_prog"*
where
"a_get_owner_document ptr = invoke (CD.a_get_owner_document_tups @ a_get_owner_document_tups) ptr ()"
end

global_interpretation *l_get_owner_document*$_{Shadow_DOM}$*_defs get_root_node_si get_root_node_si_locs*

```
  get_disconnected_nodes get_disconnected_nodes_locs get_host get_host_locs
  defines get_owner_document_tups = a_get_owner_document_tups
    and get_owner_document = a_get_owner_document
    and get_owner_document_shadow_root_ptr = a_get_owner_document_shadow_root_ptr
    and get_owner_document_tups_Core_DOM =
      "l_get_owner_document_Core_DOM_defs.a_get_owner_document_tups
        get_root_node_si get_disconnected_nodes"
    and get_owner_document_node_ptr =
      "l_get_owner_document_Core_DOM_defs.a_get_owner_document_node_ptr
        get_root_node_si get_disconnected_nodes"
  ⟨proof⟩

locale l_get_owner_document_Shadow_DOM =
  l_known_ptr known_ptr +
  l_get_owner_document_Shadow_DOM_defs get_root_node_si get_root_node_si_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_host get_host_locs +
  l_get_owner_document_defs get_owner_document +
  l_get_host get_host get_host_locs +
  CD: l_get_owner_document_Core_DOM
  get_parent get_parent_locs known_ptr_Core_DOM type_wf get_disconnected_nodes
  get_disconnected_nodes_locs get_root_node_si get_root_node_si_locs get_owner_document_Core_DOM
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
    and get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and type_wf :: "(_) heap ⇒ bool"
    and get_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_root_node_si :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_si_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_owner_document_Core_DOM :: "(_) object_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_owner_document :: "(_) object_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog" +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
  assumes get_owner_document_impl: "get_owner_document = a_get_owner_document"
begin
lemmas known_ptr_def = known_ptr_impl[unfolded a_known_ptr_def]
lemmas get_owner_document_def = a_get_owner_document_def[folded get_owner_document_impl]

lemma get_owner_document_pure [simp]:
  "pure (get_owner_document ptr) h"
⟨proof⟩

lemma get_owner_document_ptr_in_heap:
  assumes "h ⊢ ok (get_owner_document ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  ⟨proof⟩

end

interpretation i_get_owner_document?: l_get_owner_document_Shadow_DOM known_ptr DocumentClass.known_ptr
  get_parent get_parent_locs type_wf get_disconnected_nodes get_disconnected_nodes_locs
  get_root_node_si get_root_node_si_locs CD.a_get_owner_document get_host get_host_locs
  get_owner_document
  ⟨proof⟩
declare l_get_owner_document_Shadow_DOM_axioms [instances]

lemma get_owner_document_is_l_get_owner_document [instances]:
  "l_get_owner_document get_owner_document"
  ⟨proof⟩
```

remove_child

```
global_interpretation l_remove_child_Core_DOM_defs get_child_nodes get_child_nodes_locs set_child_nodes
  set_child_nodes_locs get_parent get_parent_locs get_owner_document get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs
  defines remove = "l_remove_child_Core_DOM_defs.a_remove get_child_nodes set_child_nodes get_parent
    get_owner_document get_disconnected_nodes set_disconnected_nodes"
    and remove_child = "l_remove_child_Core_DOM_defs.a_remove_child get_child_nodes set_child_nodes
    get_owner_document get_disconnected_nodes set_disconnected_nodes"
    and remove_child_locs = "l_remove_child_Core_DOM_defs.a_remove_child_locs set_child_nodes_locs
    set_disconnected_nodes_locs"
  ⟨proof⟩

interpretation i_remove_child?: l_remove_child_Core_DOM Shadow_DOM.get_child_nodes
  Shadow_DOM.get_child_nodes_locs Shadow_DOM.set_child_nodes Shadow_DOM.set_child_nodes_locs
  Shadow_DOM.get_parent Shadow_DOM.get_parent_locs Shadow_DOM.get_owner_document
  get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs remove_child remove_child_locs remove ShadowRootClass.type_wf
  ShadowRootClass.known_ptr ShadowRootClass.known_ptrs
  ⟨proof⟩
declare l_remove_child_Core_DOM_axioms [instances]
```

get_disconnected_document

```
locale l_get_disconnected_document_Core_DOM_defs =
  l_get_disconnected_nodes_defs get_disconnected_nodes get_disconnected_nodes_locs
  for get_disconnected_nodes ::
    "(_::linorder) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_get_disconnected_document :: "(_) node_ptr ⇒ (_, (_) document_ptr) dom_prog"
  where
    "a_get_disconnected_document node_ptr = do {
      check_in_heap (cast node_ptr);
      ptrs ← document_ptr_kinds_M;
      candidates ← filter_M (λdocument_ptr. do {
        disconnected_nodes ← get_disconnected_nodes document_ptr;
        return (node_ptr ∈ set disconnected_nodes)
      }) ptrs;
      (case candidates of
        Cons document_ptr [] ⇒ return document_ptr |
        _ ⇒ error HierarchyRequestError
      )
    }"
definition "a_get_disconnected_document_locs =
  (⋃document_ptr. get_disconnected_nodes_locs document_ptr) ∪
  (⋃ptr. {preserved (get_M_Object ptr RObject.nothing)})"
end

locale l_get_disconnected_document_defs =
  fixes get_disconnected_document :: "(_) node_ptr ⇒ (_, (_::linorder) document_ptr) dom_prog"
  fixes get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"

locale l_get_disconnected_document_Core_DOM =
  l_get_disconnected_document_Core_DOM_defs +
  l_get_disconnected_document_defs +
  l_get_disconnected_nodes +
  assumes get_disconnected_document_impl:
    "get_disconnected_document = a_get_disconnected_document"
  assumes get_disconnected_document_locs_impl:
    "get_disconnected_document_locs = a_get_disconnected_document_locs"
begin
lemmas get_disconnected_document_def =
```

```
  get_disconnected_document_impl[unfolded a_get_disconnected_document_def]
lemmas get_disconnected_document_locs_def =
  get_disconnected_document_locs_impl[unfolded a_get_disconnected_document_locs_def]


lemma get_disconnected_document_pure [simp]: "pure (get_disconnected_document ptr) h"
  ⟨proof⟩

lemma get_disconnected_document_ptr_in_heap [simp]:
  "h ⊢ ok (get_disconnected_document node_ptr) ⟹ node_ptr |∈| node_ptr_kinds h"
  ⟨proof⟩

lemma get_disconnected_document_disconnected_document_in_heap:
  assumes "h ⊢ get_disconnected_document child_node →r disconnected_document"
  shows "disconnected_document |∈| document_ptr_kinds h"
  ⟨proof⟩

lemma get_disconnected_document_reads:
  "reads get_disconnected_document_locs (get_disconnected_document node_ptr) h h'"
  ⟨proof⟩

end

locale l_get_disconnected_document = l_get_disconnected_document_defs +
  assumes get_disconnected_document_reads:
    "reads get_disconnected_document_locs (get_disconnected_document node_ptr) h h'"
  assumes get_disconnected_document_ptr_in_heap:
    "h ⊢ ok (get_disconnected_document node_ptr) ⟹ node_ptr |∈| node_ptr_kinds h"
  assumes get_disconnected_document_pure [simp]:
    "pure (get_disconnected_document node_ptr) h"
  assumes get_disconnected_document_disconnected_document_in_heap:
    "h ⊢ get_disconnected_document child_node →r disconnected_document ⟹
    disconnected_document |∈| document_ptr_kinds h"

global_interpretation l_get_disconnected_document_Core_DOM_defs get_disconnected_nodes
  get_disconnected_nodes_locs defines
  get_disconnected_document = a_get_disconnected_document and
  get_disconnected_document_locs = a_get_disconnected_document_locs ⟨proof⟩

interpretation i_get_disconnected_document?: l_get_disconnected_document_Core_DOM
  get_disconnected_nodes get_disconnected_nodes_locs get_disconnected_document
  get_disconnected_document_locs type_wf
  ⟨proof⟩
declare l_get_disconnected_document_Core_DOM_axioms [instances]

lemma get_disconnected_document_is_l_get_disconnected_document [instances]:
  "l_get_disconnected_document get_disconnected_document get_disconnected_document_locs"
  ⟨proof⟩
```

get_disconnected_nodes

```
locale l_set_tag_name_get_disconnected_nodes_Shadow_DOM =
  l_set_tag_name_Shadow_DOM +
  l_get_disconnected_nodes_Shadow_DOM
begin
lemma set_tag_name_get_disconnected_nodes:
  "∀w ∈ set_tag_name_locs ptr. (h ⊢ w →h h' ⟶ (∀r ∈ get_disconnected_nodes_locs ptr'. r h h'))"
  ⟨proof⟩
end

interpretation
  i_set_tag_name_get_disconnected_nodes?: l_set_tag_name_get_disconnected_nodes_Shadow_DOM type_wf
  DocumentClass.type_wf set_tag_name set_tag_name_locs get_disconnected_nodes
  get_disconnected_nodes_locs
  ⟨proof⟩
```

declare *l_set_tag_name_get_disconnected_nodes$_{Core_DOM}$_axioms[instances]*

lemma *set_tag_name_get_disconnected_nodes_is_l_set_tag_name_get_disconnected_nodes [instances]:*
"l_set_tag_name_get_disconnected_nodes type_wf set_tag_name set_tag_name_locs
get_disconnected_nodes get_disconnected_nodes_locs"
⟨*proof*⟩

adopt_node

global_interpretation *l_adopt_node$_{Core_DOM}$_defs get_owner_document get_parent get_parent_locs*
remove_child remove_child_locs get_disconnected_nodes get_disconnected_nodes_locs
set_disconnected_nodes set_disconnected_nodes_locs
defines *adopt_node = a_adopt_node*
and *adopt_node_locs = a_adopt_node_locs*
⟨*proof*⟩
interpretation *i_adopt_node?: l_adopt_node$_{Core_DOM}$ get_owner_document get_parent get_parent_locs*
remove_child remove_child_locs get_disconnected_nodes get_disconnected_nodes_locs
set_disconnected_nodes set_disconnected_nodes_locs adopt_node adopt_node_locs known_ptr type_wf
get_child_nodes get_child_nodes_locs known_ptrs set_child_nodes set_child_nodes_locs remove
⟨*proof*⟩
declare *l_adopt_node$_{Core_DOM}$_axioms[instances]*

lemma *adopt_node_is_l_adopt_node [instances]: "l_adopt_node type_wf known_ptr known_ptrs get_parent*
adopt_node adopt_node_locs get_child_nodes get_owner_document"
⟨*proof*⟩

get_shadow_root

locale *l_adopt_node_get_shadow_root$_{Shadow_DOM}$ =*
l_set_child_nodes_get_shadow_root$_{Shadow_DOM}$ +
l_set_disconnected_nodes_get_shadow_root$_{Shadow_DOM}$ +
l_adopt_node$_{Core_DOM}$
begin
lemma *adopt_node_get_shadow_root:*
"∀ w ∈ adopt_node_locs parent owner_document document_ptr.
(h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_shadow_root_locs ptr'. r h h'))"
⟨*proof*⟩
end

locale *l_adopt_node_get_shadow_root = l_adopt_node_defs + l_get_shadow_root_defs +*
assumes *adopt_node_get_shadow_root:*
"∀ w ∈ adopt_node_locs parent owner_document document_ptr.
(h ⊢ w →$_h$ h' ⟶ (∀ r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation *i_adopt_node_get_shadow_root?: l_adopt_node_get_shadow_root$_{Shadow_DOM}$ type_wf known_ptr*
DocumentClass.type_wf DocumentClass.known_ptr set_child_nodes set_child_nodes_locs
Core_DOM_Functions.set_child_nodes Core_DOM_Functions.set_child_nodes_locs get_shadow_root
get_shadow_root_locs set_disconnected_nodes set_disconnected_nodes_locs get_owner_document
get_parent get_parent_locs remove_child remove_child_locs get_disconnected_nodes
get_disconnected_nodes_locs adopt_node adopt_node_locs get_child_nodes get_child_nodes_locs
known_ptrs remove
⟨*proof*⟩
declare *l_adopt_node_get_shadow_root$_{Shadow_DOM}$_axioms[instances]*

lemma *adopt_node_get_shadow_root_is_l_adopt_node_get_shadow_root [instances]:*
"l_adopt_node_get_shadow_root adopt_node_locs get_shadow_root_locs"
⟨*proof*⟩

insert_before

global_interpretation *l_insert_before$_{Core_DOM}$_defs get_parent get_parent_locs get_child_nodes*
get_child_nodes_locs set_child_nodes set_child_nodes_locs get_ancestors_si get_ancestors_si_locs
adopt_node adopt_node_locs set_disconnected_nodes set_disconnected_nodes_locs
get_disconnected_nodes get_disconnected_nodes_locs get_owner_document
defines

```
    next_sibling = a_next_sibling and
    insert_node = a_insert_node and
    ensure_pre_insertion_validity = a_ensure_pre_insertion_validity and
    insert_before = a_insert_before and
    insert_before_locs = a_insert_before_locs
  ⟨proof⟩
global_interpretation l_append_child_Core_DOM_defs insert_before
  defines append_child = "l_append_child_Core_DOM_defs.a_append_child insert_before"
  ⟨proof⟩

interpretation i_insert_before?: l_insert_before_Core_DOM get_parent get_parent_locs get_child_nodes
  get_child_nodes_locs set_child_nodes set_child_nodes_locs get_ancestors_si get_ancestors_si_locs
  adopt_node adopt_node_locs set_disconnected_nodes set_disconnected_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs get_owner_document insert_before insert_before_locs append_child type_wf
  known_ptr known_ptrs
  ⟨proof⟩
declare l_insert_before_Core_DOM_axioms [instances]

interpretation i_append_child?: l_append_child_Core_DOM append_child insert_before insert_before_locs
  ⟨proof⟩
declare l_append_child_Core_DOM_axioms[instances]
```

get_assigned_nodes

```
fun map_filter_M2 :: "('x ⇒ ('heap, 'e, 'y option) prog) ⇒ 'x list
  ⇒ ('heap, 'e, 'y list) prog"
  where
    "map_filter_M2 f [] = return []" |
    "map_filter_M2 f (x # xs) = do {
      res ← f x;
      remainder ← map_filter_M2 f xs;
      return ((case res of  Some r ⇒ [r] | None ⇒ []) @ remainder)
    }"
lemma map_filter_M2_pure [simp]:
  assumes "⋀x. x ∈ set xs ⟹ pure (f x) h"
  shows "pure (map_filter_M2 f xs) h"
  ⟨proof⟩

lemma map_filter_pure_no_monad:
  assumes "⋀x. x ∈ set xs ⟹ pure (f x) h"
  assumes "h ⊢ map_filter_M2 f xs →_r ys"
  shows
    "ys = map the (filter (λx. x ≠ None) (map (λx. |h ⊢ f x|_r) xs))" and
    "⋀x. x ∈ set xs ⟹ h ⊢ ok (f x)"
  ⟨proof⟩

lemma map_filter_pure_foo:
  assumes "⋀x. x ∈ set xs ⟹ pure (f x) h"
  assumes "h ⊢ map_filter_M2 f xs →_r ys"
  assumes "y ∈ set ys"
  obtains x where "h ⊢ f x →_r Some y" and "x ∈ set xs"
  ⟨proof⟩


lemma map_filter_M2_in_result:
  assumes "h ⊢ map_filter_M2 P xs →_r ys"
  assumes "a ∈ set xs"
  assumes "⋀x. x ∈ set xs ⟹ pure (P x) h"
  assumes "h ⊢ P a →_r Some b"
  shows "b ∈ set ys"
  ⟨proof⟩
```

```
locale l_assigned_nodes_Shadow_DOM_defs =
  l_get_tag_name_defs get_tag_name get_tag_name_locs +
  l_get_root_node_defs get_root_node get_root_node_locs +
  l_get_host_defs get_host get_host_locs +
  l_get_child_nodes_defs get_child_nodes get_child_nodes_locs +
  l_find_slot_defs find_slot assigned_slot +
  l_remove_defs remove +
  l_insert_before_defs insert_before insert_before_locs +
  l_append_child_defs append_child +
  l_remove_shadow_root_defs remove_shadow_root remove_shadow_root_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and find_slot :: "bool ⇒ (_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and assigned_slot :: "(_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and remove :: "(_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and insert_before ::
    "(_) object_ptr ⇒ (_) node_ptr ⇒ (_) node_ptr option ⇒ ((_) heap, exception, unit) prog"
    and insert_before_locs ::  "(_) object_ptr ⇒ (_) object_ptr option ⇒ (_) document_ptr ⇒
    (_) document_ptr ⇒ (_, unit) dom_prog set"
    and append_child :: "(_) object_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root_locs :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒
    ((_) heap, exception, unit) prog set"
begin
definition a_assigned_nodes :: "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"
  where
    "a_assigned_nodes slot = do {
      tag ← get_tag_name slot;
      (if tag ≠ ''slot''
      then error HierarchyRequestError
      else return ());
      root ← get_root_node (cast slot);
      if is_shadow_root_ptr_kind root
      then do {
        host ← get_host (the (cast root));
        children ← get_child_nodes (cast host);
        filter_M (λslotable. do {
          found_slot ← find_slot False slotable;
          return (found_slot = Some slot)}) children}
      else return []}"

partial_function (dom_prog) a_assigned_nodes_flatten ::
  "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"
  where
    "a_assigned_nodes_flatten slot = do {
      tag ← get_tag_name slot;
      (if tag ≠ ''slot''
      then error HierarchyRequestError
      else return ());
      root ← get_root_node (cast slot);
      (if is_shadow_root_ptr_kind root
      then do {
        slotables ← a_assigned_nodes slot;
        slotables_or_child_nodes ← (if slotables = []
        then do {
          get_child_nodes (cast slot)
        } else do {
```

```
        return slotables
      });
      list_of_lists ← map_M (λnode_ptr. do {
        (case cast node_ptr of
          Some element_ptr ⇒ do {
            tag ← get_tag_name element_ptr;
            (if tag = ''slot''
            then do {
              root ← get_root_node (cast element_ptr);
              (if is_shadow_root_ptr_kind root
              then do {
                a_assigned_nodes_flatten element_ptr
              } else do {
                return [node_ptr]
              })
            } else do {
              return [node_ptr]
            })
          }
          | None ⇒ return [node_ptr])
      }) slotables_or_child_nodes;
      return (concat list_of_lists)
    } else return [])
  }"

definition a_flatten_dom :: "(_, unit) dom_prog" where
  "a_flatten_dom = do {
    tups ← element_ptr_kinds_M ⤜ map_filter_M2 (λelement_ptr. do {
      tag ← get_tag_name element_ptr;
      assigned_nodes ← a_assigned_nodes element_ptr;
      (if tag = ''slot'' ∧ assigned_nodes ≠ []
       then return (Some (element_ptr, assigned_nodes))
       else return None)});
    forall_M (λ(slot, assigned_nodes). do {
      get_child_nodes (cast slot) ⤜ forall_M remove;
      forall_M (append_child (cast slot)) assigned_nodes
    }) tups;
    shadow_root_ptr_kinds_M ⤜ 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
    });
    return ()
  }"
end

global_interpretation l_assigned_nodes_Shadow_DOM_defs get_child_nodes get_child_nodes_locs
  get_tag_name get_tag_name_locs get_root_node get_root_node_locs get_host get_host_locs find_slot
  assigned_slot remove insert_before insert_before_locs append_child remove_shadow_root
  remove_shadow_root_locs
  defines assigned_nodes = a_assigned_nodes
    and assigned_nodes_flatten = a_assigned_nodes_flatten
    and flatten_dom = a_flatten_dom
  ⟨proof⟩

declare a_assigned_nodes_flatten.simps [code]

locale l_assigned_nodes_defs =
  fixes assigned_nodes :: "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"
  fixes assigned_nodes_flatten :: "(_) element_ptr ⇒ (_, (_) node_ptr list) dom_prog"
  fixes flatten_dom :: "(_, unit) dom_prog"
```

```
locale l_assigned_nodes_Shadow_DOM =
  l_assigned_nodes_defs
  assigned_nodes assigned_nodes_flatten flatten_dom
  + l_assigned_nodes_Shadow_DOM_defs
  get_child_nodes get_child_nodes_locs get_tag_name get_tag_name_locs get_root_node
  get_root_node_locs get_host get_host_locs find_slot assigned_slot remove insert_before
  insert_before_locs append_child remove_shadow_root remove_shadow_root_locs
  + l_get_shadow_root
  type_wf get_shadow_root get_shadow_root_locs
  + l_set_shadow_root
  type_wf set_shadow_root set_shadow_root_locs
  + l_remove
  + l_insert_before
  insert_before insert_before_locs
  + l_find_slot
  find_slot assigned_slot
  + l_get_tag_name
  type_wf get_tag_name get_tag_name_locs
  + l_get_root_node
  get_root_node get_root_node_locs get_parent get_parent_locs
  + l_get_host
  get_host get_host_locs
  + l_get_child_nodes
  type_wf known_ptr get_child_nodes get_child_nodes_locs
  + l_to_tree_order
  to_tree_order
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and assigned_nodes :: "(_) element_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and assigned_nodes_flatten :: "(_) element_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and flatten_dom :: "((_) heap, exception, unit) prog"
    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_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_root_node :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr) prog"
    and get_root_node_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and find_slot :: "bool ⇒ (_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and assigned_slot :: "(_) node_ptr ⇒ ((_) heap, exception, (_) element_ptr option) prog"
    and remove :: "(_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and insert_before ::
    "(_) object_ptr ⇒ (_) node_ptr ⇒ (_) node_ptr option ⇒ ((_) heap, exception, unit) prog"
    and insert_before_locs ::  "(_) object_ptr ⇒ (_) object_ptr option ⇒ (_) document_ptr ⇒
    (_) document_ptr ⇒ (_, unit) dom_prog set"
    and append_child :: "(_) object_ptr ⇒ (_) node_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, unit) prog"
    and remove_shadow_root_locs :: "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒
    ((_) heap, exception, unit) prog set"
    and type_wf :: "(_) heap ⇒ bool"
    and get_shadow_root ::
    "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    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 get_parent :: "(_) node_ptr ⇒ ((_) heap, exception, (_) object_ptr option) prog"
    and get_parent_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and to_tree_order :: "(_) object_ptr ⇒ ((_) heap, exception, (_) object_ptr list) prog" +
  assumes assigned_nodes_impl: "assigned_nodes = a_assigned_nodes"
  assumes flatten_dom_impl: "flatten_dom = a_flatten_dom"
begin
lemmas assigned_nodes_def = assigned_nodes_impl[unfolded a_assigned_nodes_def]
```

lemmas *flatten_dom_def = flatten_dom_impl[unfolded a_flatten_dom_def, folded assigned_nodes_impl]*

lemma *assigned_nodes_pure [simp]: "pure (assigned_nodes slot) h"*
⟨*proof*⟩

lemma *assigned_nodes_ptr_in_heap:*
assumes *"h ⊢ ok (assigned_nodes slot)"*
shows *"slot |∈| element_ptr_kinds h"*
⟨*proof*⟩

lemma *assigned_nodes_slot_is_slot:*
assumes *"h ⊢ ok (assigned_nodes slot)"*
shows *"h ⊢ get_tag_name slot →$_r$ ''slot''"*
⟨*proof*⟩

lemma *assigned_nodes_different_ptr:*
assumes *"h ⊢ assigned_nodes slot →$_r$ nodes"*
assumes *"h ⊢ assigned_nodes slot' →$_r$ nodes'"*
assumes *"slot ≠ slot'"*
shows *"set nodes ∩ set nodes' = {}"*
⟨*proof*⟩
end

interpretation *i_assigned_nodes?: l_assigned_nodes$_{Shadow_DOM}$ known_ptr assigned_nodes*
assigned_nodes_flatten flatten_dom get_child_nodes get_child_nodes_locs get_tag_name
get_tag_name_locs get_root_node get_root_node_locs get_host get_host_locs find_slot
assigned_slot remove insert_before insert_before_locs append_child remove_shadow_root
remove_shadow_root_locs type_wf get_shadow_root get_shadow_root_locs set_shadow_root
set_shadow_root_locs get_parent get_parent_locs to_tree_order
⟨*proof*⟩
declare *l_assigned_nodes$_{Shadow_DOM}$_axioms [instances]*

locale *l_assigned_nodes = l_assigned_nodes_defs +*
assumes *assigned_nodes_pure [simp]: "pure (assigned_nodes slot) h"*
assumes *assigned_nodes_ptr_in_heap:*
"h ⊢ ok (assigned_nodes slot) ⟹ slot |∈| element_ptr_kinds h"
assumes *assigned_nodes_slot_is_slot:*
"h ⊢ ok (assigned_nodes slot) ⟹ h ⊢ get_tag_name slot →$_r$ ''slot''"
assumes *assigned_nodes_different_ptr:*
"h ⊢ assigned_nodes slot →$_r$ nodes ⟹ h ⊢ assigned_nodes slot' →$_r$ nodes' ⟹
slot ≠ slot' ⟹ set nodes ∩ set nodes' = {}"

lemma *assigned_nodes_is_l_assigned_nodes [instances]: "l_assigned_nodes assigned_nodes"*
⟨*proof*⟩

set_val

locale *l_set_val$_{Shadow_DOM}$ =*
CD: l_set_val$_{Core_DOM}$ type_wf$_{Core_DOM}$ set_val set_val_locs +
l_type_wf type_wf
for *type_wf :: "(_) heap ⇒ bool"*
and *type_wf$_{Core_DOM}$ :: "(_) heap ⇒ bool"*
and *set_val :: "(_) character_data_ptr ⇒ char list ⇒ (_, unit) dom_prog"*
and *set_val_locs :: "(_) character_data_ptr ⇒ (_, unit) dom_prog set" +*
assumes *type_wf_impl: "type_wf = ShadowRootClass.type_wf"*
begin

lemma *set_val_ok:*
"type_wf h ⟹ character_data_ptr |∈| character_data_ptr_kinds h ⟹
h ⊢ ok (set_val character_data_ptr tag)"
⟨*proof*⟩

lemma *set_val_writes:*

```
  "writes (set_val_locs character_data_ptr) (set_val character_data_ptr tag) h h'"
  ⟨proof⟩

lemma set_val_pointers_preserved:
  assumes "w ∈ set_val_locs character_data_ptr"
  assumes "h ⊢ w →_h h'"
  shows "object_ptr_kinds h = object_ptr_kinds h'"
  ⟨proof⟩

lemma set_val_typess_preserved:
  assumes "w ∈ set_val_locs character_data_ptr"
  assumes "h ⊢ w →_h h'"
  shows "type_wf h = type_wf h'"
  ⟨proof⟩
end

interpretation
  i_set_val?: l_set_val_Shadow_DOM type_wf DocumentClass.type_wf set_val set_val_locs
  ⟨proof⟩
declare l_set_val_Shadow_DOM_axioms[instances]

lemma set_val_is_l_set_val [instances]: "l_set_val type_wf set_val set_val_locs"
  ⟨proof⟩
```

get_shadow_root locale `l_set_val_get_shadow_root_Shadow_DOM =`

```
  l_set_val_Shadow_DOM +
  l_get_shadow_root_Shadow_DOM
begin
lemma set_val_get_shadow_root:
  "∀w ∈ set_val_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"
  ⟨proof⟩
end

locale l_set_val_get_shadow_root = l_set_val + l_get_shadow_root +
  assumes set_val_get_shadow_root:
    "∀w ∈ set_val_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_shadow_root_locs ptr'. r h h'))"

interpretation
  i_set_val_get_shadow_root?: l_set_val_get_shadow_root_Shadow_DOM type_wf DocumentClass.type_wf
  set_val set_val_locs
  get_shadow_root get_shadow_root_locs
  ⟨proof⟩
declare l_set_val_get_shadow_root_Shadow_DOM_axioms[instances]

lemma set_val_get_shadow_root_is_l_set_val_get_shadow_root [instances]:
  "l_set_val_get_shadow_root type_wf set_val set_val_locs get_shadow_root
                                get_shadow_root_locs"
  ⟨proof⟩
```

get_tag_type locale `l_set_val_get_tag_name_Shadow_DOM =`

```
  l_set_val_Shadow_DOM +
  l_get_tag_name_Shadow_DOM
begin
lemma set_val_get_tag_name:
  "∀w ∈ set_val_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"
  ⟨proof⟩
end

locale l_set_val_get_tag_name = l_set_val + l_get_tag_name +
  assumes set_val_get_tag_name:
    "∀w ∈ set_val_locs ptr. (h ⊢ w →_h h' ⟶ (∀r ∈ get_tag_name_locs ptr'. r h h'))"

interpretation
```

```
  i_set_val_get_tag_name?: l_set_val_get_tag_name_Shadow_DOM type_wf DocumentClass.type_wf set_val
  set_val_locs get_tag_name get_tag_name_locs
  ⟨proof⟩
declare l_set_val_get_tag_name_Shadow_DOM_axioms[instances]

lemma set_val_get_tag_name_is_l_set_val_get_tag_name [instances]:
  "l_set_val_get_tag_name type_wf set_val set_val_locs get_tag_name get_tag_name_locs"
  ⟨proof⟩
```

create_character_data

```
locale l_create_character_data_Shadow_DOM =
  CD: l_create_character_data_Core_DOM _ _ _ _ _ _ type_wf_Core_DOM _ known_ptr_Core_DOM +
  l_known_ptr known_ptr
  for known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool" +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
begin

lemma create_character_data_document_in_heap:
  assumes "h ⊢ ok (create_character_data document_ptr text)"
  shows "document_ptr |∈| document_ptr_kinds h"
  ⟨proof⟩

lemma create_character_data_known_ptr:
  assumes "h ⊢ create_character_data document_ptr text →_r new_character_data_ptr"
  shows "known_ptr (cast new_character_data_ptr)"
  ⟨proof⟩
end

locale l_create_character_data = l_create_character_data_defs

interpretation i_create_character_data?: l_create_character_data_Shadow_DOM get_disconnected_nodes
  get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs set_val set_val_locs
  create_character_data known_ptr DocumentClass.type_wf DocumentClass.known_ptr
  ⟨proof⟩
declare l_create_character_data_Core_DOM_axioms [instances]
```

create_element

```
locale l_create_element_Shadow_DOM =
  CD: 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_Core_DOM create_element
  known_ptr_Core_DOM +
  l_known_ptr known_ptr
  for 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 type_wf :: "(_) heap ⇒ bool"
    and create_element ::
    "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    and known_ptr_Core_DOM :: "(_) object_ptr ⇒ bool" +
  assumes known_ptr_impl: "known_ptr = a_known_ptr"
begin
lemmas create_element_def = CD.create_element_def
```

```
lemma create_element_document_in_heap:
  assumes "h ⊢ ok (create_element document_ptr tag)"
  shows "document_ptr |∈| document_ptr_kinds h"
  ⟨proof⟩

lemma create_element_known_ptr:
  assumes "h ⊢ create_element document_ptr tag →_r new_element_ptr"
  shows "known_ptr (cast new_element_ptr)"
⟨proof⟩
end

interpretation i_create_element?: l_create_element_Shadow_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 DocumentClass.type_wf DocumentClass.known_ptr
  ⟨proof⟩
declare l_create_element_Shadow_DOM_axioms[instances]
```

2.3.2 A wellformed heap (Core DOM)

wellformed_heap

```
locale l_heap_is_wellformed_Shadow_DOM_defs =
  CD: l_heap_is_wellformed_Core_DOM_defs get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs +
  l_get_shadow_root_defs get_shadow_root get_shadow_root_locs +
  l_get_tag_name_defs get_tag_name get_tag_name_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) 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 get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
begin
definition a_host_shadow_root_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
  where
    "a_host_shadow_root_rel h = (λ(x, y). (cast x, cast y)) ` {(host, shadow_root).
      host |∈| element_ptr_kinds h ∧ |h ⊢ get_shadow_root host|_r = Some shadow_root}"

lemma a_host_shadow_root_rel_code [code]: "a_host_shadow_root_rel h = set (concat (map
  (λhost. (case |h ⊢ get_shadow_root host|_r of
    Some shadow_root ⇒ [(cast host, cast shadow_root)] |
    None ⇒ []))
  (sorted_list_of_fset (element_ptr_kinds h)))
)"
  ⟨proof⟩

definition a_all_ptrs_in_heap :: "(_) heap ⇒ bool" where
  "a_all_ptrs_in_heap h = ((∀host shadow_root_ptr.
    (h ⊢ get_shadow_root host →_r Some shadow_root_ptr) ⟶
      shadow_root_ptr |∈| shadow_root_ptr_kinds h))"

definition a_distinct_lists :: "(_) heap ⇒ bool"
  where
    "a_distinct_lists h = distinct (concat (
      map (λelement_ptr. (case |h ⊢ get_shadow_root element_ptr|_r of
        Some shadow_root_ptr ⇒ [shadow_root_ptr] | None ⇒ []))
        |h ⊢ element_ptr_kinds_M|_r
    ))"


definition a_shadow_root_valid :: "(_) heap ⇒ bool" where
```

```
  "a_shadow_root_valid h = (∀shadow_root_ptr ∈ fset (shadow_root_ptr_kinds h).
     (∃host ∈ fset(element_ptr_kinds h).
       |h ⊢ get_tag_name host|_r ∈ safe_shadow_root_element_types ∧
       |h ⊢ get_shadow_root host|_r = Some shadow_root_ptr))"

definition a_heap_is_wellformed :: "(_) heap ⇒ bool"
  where
    "a_heap_is_wellformed h ⟷ CD.a_heap_is_wellformed h ∧
     acyclic (CD.a_parent_child_rel h ∪ a_host_shadow_root_rel h) ∧
     a_all_ptrs_in_heap h ∧
     a_distinct_lists h ∧
     a_shadow_root_valid h"
end

global_interpretation l_heap_is_wellformed_Shadow_DOM_defs 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
  defines heap_is_wellformed = a_heap_is_wellformed
    and parent_child_rel = CD.a_parent_child_rel
    and host_shadow_root_rel = a_host_shadow_root_rel
    and all_ptrs_in_heap = a_all_ptrs_in_heap
    and distinct_lists = a_distinct_lists
    and shadow_root_valid = a_shadow_root_valid
    and heap_is_wellformed_Core_DOM = CD.a_heap_is_wellformed
    and parent_child_rel_Core_DOM = CD.a_parent_child_rel
    and acyclic_heap_Core_DOM = CD.a_acyclic_heap
    and all_ptrs_in_heap_Core_DOM = CD.a_all_ptrs_in_heap
    and distinct_lists_Core_DOM = CD.a_distinct_lists
    and owner_document_valid_Core_DOM = CD.a_owner_document_valid
  ⟨proof⟩

interpretation i_heap_is_wellformed_Core_DOM: l_heap_is_wellformed_Core_DOM known_ptr type_wf
  get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs
  heap_is_wellformed_Core_DOM parent_child_rel
  ⟨proof⟩
declare i_heap_is_wellformed_Core_DOM.l_heap_is_wellformed_Core_DOM_axioms[instances]

lemma heap_is_wellformed_Core_DOM_is_l_heap_is_wellformed [instances]:
  "l_heap_is_wellformed type_wf known_ptr heap_is_wellformed_Core_DOM parent_child_rel get_child_nodes
  get_disconnected_nodes"
  ⟨proof⟩


locale l_heap_is_wellformed_Shadow_DOM =
  l_heap_is_wellformed_Shadow_DOM_defs
  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
  + CD: l_heap_is_wellformed_Core_DOM
  known_ptr type_wf get_child_nodes get_child_nodes_locs get_disconnected_nodes
  get_disconnected_nodes_locs heap_is_wellformed_Core_DOM parent_child_rel
  + l_heap_is_wellformed_defs
  heap_is_wellformed parent_child_rel
  + l_get_host_Shadow_DOM
  get_shadow_root get_shadow_root_locs get_host get_host_locs type_wf
  + l_get_disconnected_document_Core_DOM get_disconnected_nodes get_disconnected_nodes_locs
  get_disconnected_document get_disconnected_document_locs type_wf
  + l_get_shadow_root_Shadow_DOM type_wf get_shadow_root get_shadow_root_locs
  for get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) 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 get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
```

```
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and known_ptr :: "(_) object_ptr ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set" +
  assumes heap_is_wellformed_impl: "heap_is_wellformed = a_heap_is_wellformed"
begin
lemmas heap_is_wellformed_def =
  heap_is_wellformed_impl[unfolded a_heap_is_wellformed_def,
    folded CD.heap_is_wellformed_impl CD.parent_child_rel_impl]

lemma a_distinct_lists_code [code]:
  "a_all_ptrs_in_heap h = ((∀host ∈ fset (element_ptr_kinds h).
    h ⊢ ok (get_shadow_root host) ⟶ (case |h ⊢ get_shadow_root host|_r of
    Some shadow_root_ptr ⇒ shadow_root_ptr |∈| shadow_root_ptr_kinds h |
    None ⇒ True)))"
  ⟨proof⟩

lemma get_shadow_root_shadow_root_ptr_in_heap:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  ⟨proof⟩

lemma get_host_ptr_in_heap:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ get_host shadow_root_ptr →_r host"
  shows "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  ⟨proof⟩

lemma shadow_root_same_host:
  assumes "heap_is_wellformed h" and "type_wf h"
  assumes "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
  assumes "h ⊢ get_shadow_root host' →_r Some shadow_root_ptr"
  shows "host = host'"
⟨proof⟩

lemma shadow_root_host_dual:
  assumes "h ⊢ get_host shadow_root_ptr →_r host"
  shows "h ⊢ get_shadow_root host →_r Some shadow_root_ptr"
  ⟨proof⟩

lemma disc_doc_disc_node_dual:
  assumes "h ⊢ get_disconnected_document disc_node →_r disc_doc"
  obtains disc_nodes where
    "h ⊢ get_disconnected_nodes disc_doc →_r disc_nodes" and
    "disc_node ∈ set disc_nodes"
  ⟨proof⟩

lemma get_host_valid_tag_name:
  assumes "heap_is_wellformed h" and "type_wf h"
  assumes "h ⊢ get_host shadow_root_ptr →_r host"
  assumes "h ⊢ get_tag_name host →_r tag"
  shows "tag ∈ safe_shadow_root_element_types"
⟨proof⟩
```

lemma *a_host_shadow_root_rel_finite: "finite (a_host_shadow_root_rel h)"*
⟨*proof*⟩

lemma *heap_is_wellformed_children_in_heap:*
"heap_is_wellformed h ⟹ h ⊢ get_child_nodes ptr →r children ⟹ child ∈ set children ⟹
child |∈| node_ptr_kinds h"
⟨*proof*⟩
lemma *heap_is_wellformed_disc_nodes_in_heap:*
"heap_is_wellformed h ⟹ h ⊢ get_disconnected_nodes document_ptr →r disc_nodes ⟹
node ∈ set disc_nodes ⟹ node |∈| node_ptr_kinds h"
⟨*proof*⟩
lemma *heap_is_wellformed_one_parent:*
"heap_is_wellformed h ⟹ h ⊢ get_child_nodes ptr →r children ⟹
h ⊢ get_child_nodes ptr' →r children' ⟹ set children ∩ set children' ≠ {} ⟹ ptr = ptr'"
⟨*proof*⟩
lemma *heap_is_wellformed_one_disc_parent:*
"heap_is_wellformed h ⟹ h ⊢ get_disconnected_nodes document_ptr →r disc_nodes ⟹
h ⊢ get_disconnected_nodes document_ptr' →r disc_nodes' ⟹ set disc_nodes ∩ set disc_nodes' ≠ {}
⟹ document_ptr = document_ptr'"
⟨*proof*⟩
lemma *heap_is_wellformed_children_disc_nodes_different:*
"heap_is_wellformed h ⟹ h ⊢ get_child_nodes ptr →r children ⟹
h ⊢ get_disconnected_nodes document_ptr →r disc_nodes ⟹ set children ∩ set disc_nodes = {}"
⟨*proof*⟩
lemma *heap_is_wellformed_disconnected_nodes_distinct:*
"heap_is_wellformed h ⟹ h ⊢ get_disconnected_nodes document_ptr →r disc_nodes ⟹
distinct disc_nodes"
⟨*proof*⟩
lemma *heap_is_wellformed_children_distinct:*
"heap_is_wellformed h ⟹ h ⊢ get_child_nodes ptr →r children ⟹ distinct children"
⟨*proof*⟩
lemma *heap_is_wellformed_children_disc_nodes:*
"heap_is_wellformed h ⟹ node_ptr |∈| node_ptr_kinds h ⟹
¬(∃parent ∈ fset (object_ptr_kinds h). node_ptr ∈ set |h ⊢ get_child_nodes parent|r) ⟹
(∃document_ptr ∈ fset (document_ptr_kinds h).
node_ptr ∈ set |h ⊢ get_disconnected_nodes document_ptr|r)"
⟨*proof*⟩
lemma *parent_child_rel_finite: "heap_is_wellformed h ⟹ finite (parent_child_rel h)"*
⟨*proof*⟩
lemma *parent_child_rel_acyclic: "heap_is_wellformed h ⟹ acyclic (parent_child_rel h)"*
⟨*proof*⟩
lemma *parent_child_rel_child_in_heap:*
"heap_is_wellformed h ⟹ type_wf h ⟹ known_ptr parent ⟹
(parent, child_ptr) ∈ parent_child_rel h ⟹ child_ptr |∈| object_ptr_kinds h"
⟨*proof*⟩
end

interpretation *i_heap_is_wellformed?: l_heap_is_wellformed*$_{Shadow_DOM}$ *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
⟨*proof*⟩
declare *l_heap_is_wellformed*$_{Shadow_DOM}$*_axioms [instances]*

lemma *heap_is_wellformed_is_l_heap_is_wellformed [instances]:*
"l_heap_is_wellformed ShadowRootClass.type_wf ShadowRootClass.known_ptr
Shadow_DOM.heap_is_wellformed Shadow_DOM.parent_child_rel Shadow_DOM.get_child_nodes
get_disconnected_nodes"
⟨*proof*⟩

get_parent

interpretation *i_get_parent_wf$_{Core_DOM}$: l_get_parent_wf$_{Core_DOM}$ known_ptr type_wf get_child_nodes get_child_nodes_locs known_ptrs get_parent get_parent_locs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_disconnected_nodes*
⟨*proof*⟩
declare *i_get_parent_wf$_{Core_DOM}$.l_get_parent_wf$_{Core_DOM}$_axioms[instances]*

interpretation *i_get_parent_wf2$_{Core_DOM}$: l_get_parent_wf2$_{Core_DOM}$ known_ptr type_wf get_child_nodes get_child_nodes_locs known_ptrs get_parent get_parent_locs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_disconnected_nodes get_disconnected_nodes_locs*
⟨*proof*⟩
declare *i_get_parent_wf2$_{Core_DOM}$.l_get_parent_wf2$_{Core_DOM}$_axioms[instances]*

lemma *get_parent_wf$_{Core_DOM}$_is_l_get_parent_wf [instances]: "l_get_parent_wf type_wf known_ptr known_ptrs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_child_nodes get_parent"*
⟨*proof*⟩

get_disconnected_nodes

set_disconnected_nodes

get_disconnected_nodes **interpretation** *i_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$: l_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$ known_ptr type_wf get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_child_nodes*
⟨*proof*⟩
declare *i_set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$.l_set_disconnected_nodes_get_disconnected_node*

lemma *set_disconnected_nodes_get_disconnected_nodes_wf$_{Core_DOM}$_is_l_set_disconnected_nodes_get_disconnected_node [instances]:*
"l_set_disconnected_nodes_get_disconnected_nodes_wf type_wf known_ptr heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_child_nodes get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs"
⟨*proof*⟩

get_root_node **interpretation** *i_get_root_node_wf$_{Core_DOM}$:l_get_root_node_wf$_{Core_DOM}$ known_ptr type_wf known_ptrs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_parent get_parent_locs get_ancestors get_ancestors_locs get_root_node get_root_node_locs*
⟨*proof*⟩
declare *i_get_root_node_wf$_{Core_DOM}$.l_get_root_node_wf$_{Core_DOM}$_axioms[instances]*

lemma *get_ancestors_wf$_{Core_DOM}$_is_l_get_ancestors_wf [instances]:*
"l_get_ancestors_wf heap_is_wellformed$_{Core_DOM}$ parent_child_rel known_ptr known_ptrs type_wf get_ancestors get_ancestors_locs get_child_nodes get_parent"
⟨*proof*⟩

lemma *get_root_node_wf$_{Core_DOM}$_is_l_get_root_node_wf [instances]:*
"l_get_root_node_wf heap_is_wellformed$_{Core_DOM}$ get_root_node type_wf known_ptr known_ptrs get_ancestors get_parent"
⟨*proof*⟩

to_tree_order

interpretation *i_to_tree_order_wf$_{Core_DOM}$: l_to_tree_order_wf$_{Core_DOM}$ known_ptr type_wf get_child_nodes get_child_nodes_locs to_tree_order known_ptrs get_parent get_parent_locs heap_is_wellformed$_{Core_DOM}$ parent_child_rel get_disconnected_nodes get_disconnected_nodes_locs*
⟨*proof*⟩
declare *i_to_tree_order_wf$_{Core_DOM}$.l_to_tree_order_wf$_{Core_DOM}$_axioms [instances]*

lemma *to_tree_order_wf$_{Core_DOM}$_is_l_to_tree_order_wf [instances]:*
"l_to_tree_order_wf heap_is_wellformed$_{Core_DOM}$ parent_child_rel type_wf known_ptr known_ptrs

```
  to_tree_order get_parent get_child_nodes"
  ⟨proof⟩

get_root_node interpretation i_to_tree_order_wf_get_root_node_wf_Core_DOM: l_to_tree_order_wf_get_root_node_wf_C
  known_ptr type_wf known_ptrs heap_is_wellformed_Core_DOM parent_child_rel get_child_nodes
  get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_parent get_parent_locs
  get_ancestors get_ancestors_locs get_root_node get_root_node_locs to_tree_order
  ⟨proof⟩
declare i_to_tree_order_wf_get_root_node_wf_Core_DOM.l_to_tree_order_wf_get_root_node_wf_Core_DOM_axioms
[instances]

lemma to_tree_order_wf_get_root_node_wf_Core_DOM_is_l_to_tree_order_wf_get_root_node_wf [instances]:
  "l_to_tree_order_wf_get_root_node_wf type_wf known_ptr known_ptrs to_tree_order get_root_node
    heap_is_wellformed_Core_DOM"
  ⟨proof⟩
```

remove_child

```
interpretation i_remove_child_wf2_Core_DOM: l_remove_child_wf2_Core_DOM get_child_nodes get_child_nodes_locs
  set_child_nodes set_child_nodes_locs get_parent
  get_parent_locs get_owner_document get_disconnected_nodes get_disconnected_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs remove_child remove_child_locs remove type_wf
  known_ptr known_ptrs heap_is_wellformed_Core_DOM parent_child_rel
  ⟨proof⟩
declare i_remove_child_wf2_Core_DOM.l_remove_child_wf2_Core_DOM_axioms [instances]

lemma remove_child_wf2_Core_DOM_is_l_remove_child_wf2 [instances]: "l_remove_child_wf2 type_wf known_ptr
  known_ptrs remove_child heap_is_wellformed_Core_DOM get_child_nodes remove"
  ⟨proof⟩
```

2.3.3 A wellformed heap

get_parent

```
interpretation i_get_parent_wf?: l_get_parent_wf_Core_DOM known_ptr type_wf get_child_nodes
  get_child_nodes_locs known_ptrs get_parent get_parent_locs heap_is_wellformed parent_child_rel
  get_disconnected_nodes
  ⟨proof⟩
declare l_get_parent_wf_Core_DOM_axioms[instances]

lemma get_parent_wf_is_l_get_parent_wf [instances]:
  "l_get_parent_wf ShadowRootClass.type_wf ShadowRootClass.known_ptr ShadowRootClass.known_ptrs
  heap_is_wellformed parent_child_rel Shadow_DOM.get_child_nodes Shadow_DOM.get_parent"
  ⟨proof⟩
```

remove_shadow_root

```
locale l_remove_shadow_root_wf_Shadow_DOM =
  l_get_tag_name +
  l_get_disconnected_nodes  +
  l_set_shadow_root_get_tag_name +
  l_get_child_nodes  +
  l_heap_is_wellformed_Shadow_DOM +
  l_remove_shadow_root_Shadow_DOM +
  l_delete_shadow_root_get_disconnected_nodes +
  l_delete_shadow_root_get_child_nodes +
  l_set_shadow_root_get_disconnected_nodes +
  l_set_shadow_root_get_child_nodes +
  l_delete_shadow_root_get_tag_name +
  l_set_shadow_root_get_shadow_root +
  l_delete_shadow_root_get_shadow_root +
  l_get_parent_Core_DOM
begin
```

lemma *remove_shadow_root_preserves:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ remove_shadow_root ptr →$_h$ h'"*
shows *"known_ptrs h'"* **and** *"type_wf h'" "heap_is_wellformed h'"*
⟨*proof*⟩
end

interpretation *i_remove_shadow_root_wf?: l_remove_shadow_root_wf$_{Shadow_DOM}$*
type_wf get_tag_name get_tag_name_locs get_disconnected_nodes get_disconnected_nodes_locs
set_shadow_root set_shadow_root_locs known_ptr get_child_nodes get_child_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 remove_shadow_root
remove_shadow_root_locs known_ptrs get_parent get_parent_locs
⟨*proof*⟩
declare *l_remove_shadow_root_wf$_{Shadow_DOM}$_axioms [instances]*

get_root_node

interpretation *i_get_root_node_wf?: l_get_root_node_wf$_{Core_DOM}$ known_ptr type_wf known_ptrs*
heap_is_wellformed parent_child_rel get_child_nodes get_child_nodes_locs get_disconnected_nodes
get_disconnected_nodes_locs get_parent get_parent_locs get_ancestors get_ancestors_locs
get_root_node get_root_node_locs
⟨*proof*⟩
declare *l_get_root_node_wf$_{Core_DOM}$_axioms[instances]*

lemma *get_ancestors_wf_is_l_get_ancestors_wf [instances]:*
"l_get_ancestors_wf heap_is_wellformed parent_child_rel known_ptr known_ptrs type_wf get_ancestors
get_ancestors_locs get_child_nodes get_parent"
⟨*proof*⟩

lemma *get_root_node_wf_is_l_get_root_node_wf [instances]:*
"l_get_root_node_wf heap_is_wellformed get_root_node type_wf known_ptr known_ptrs get_ancestors
get_parent"
⟨*proof*⟩

get_parent_get_host

locale *l_get_parent_get_host_wf$_{Shadow_DOM}$ =*
l_heap_is_wellformed$_{Shadow_DOM}$ +
l_get_parent_wf +
l_get_shadow_root +
l_get_host +
l_get_child_nodes
begin
lemma *host_shadow_root_rel_finite: "finite (a_host_shadow_root_rel h)"*
⟨*proof*⟩

lemma *host_shadow_root_rel_shadow_root:*
"h ⊢ get_shadow_root host →$_r$ shadow_root_option ⟹
shadow_root_option = Some shadow_root ⟷ ((cast host, cast shadow_root) ∈ a_host_shadow_root_rel h)"
⟨*proof*⟩

lemma *host_shadow_root_rel_host:*
"heap_is_wellformed h ⟹ h ⊢ get_host shadow_root →$_r$ host ⟹
(cast host, cast shadow_root) ∈ a_host_shadow_root_rel h"
⟨*proof*⟩

lemma *heap_wellformed_induct_si [consumes 1, case_names step]:*
assumes *"heap_is_wellformed h"*
assumes *"⋀parent. (⋀children child. h ⊢ get_child_nodes parent →$_r$ children ⟹*
child ∈ set children ⟹ P (cast child)) ⟹ (⋀shadow_root host. parent = cast host ⟹
h ⊢ get_shadow_root host →$_r$ Some shadow_root ⟹ P (cast shadow_root)) ⟹ P parent"
shows *"P ptr"*

⟨*proof*⟩

lemma *heap_wellformed_induct_rev_si [consumes 1, case_names step]:*
assumes *"heap_is_wellformed h"*
assumes *"⋀child. (⋀parent child_node. child = cast child_node ⟹ h ⊢ get_parent child_node →r Some parent ⟹ P parent) ⟹ (⋀host shadow_root. child = cast shadow_root ⟹ h ⊢ get_host shadow_root →r host ⟹ P (cast host)) ⟹ P child"*
shows *"P ptr"*
⟨*proof*⟩
end

interpretation *i_get_parent_get_host_wf?: l_get_parent_get_host_wf*$_{Shadow_DOM}$
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 get_parent get_parent_locs*
⟨*proof*⟩
declare *l_get_parent_get_host_wf*$_{Shadow_DOM}$*_axioms [instances]*

locale *l_get_parent_get_host_wf =*
l_heap_is_wellformed_defs +
l_get_parent_defs +
l_get_shadow_root_defs +
l_get_host_defs +
l_get_child_nodes_defs +
assumes *heap_wellformed_induct_si [consumes 1, case_names step]:*
"heap_is_wellformed h ⟹ (⋀parent. (⋀children child. h ⊢ get_child_nodes parent →r children ⟹ child ∈ set children ⟹ P (cast child)) ⟹ (⋀shadow_root host. parent = cast host ⟹ h ⊢ get_shadow_root host →r Some shadow_root ⟹ P (cast shadow_root)) ⟹ P parent) ⟹ P ptr"
assumes *heap_wellformed_induct_rev_si [consumes 1, case_names step]:*
"heap_is_wellformed h ⟹ (⋀child. (⋀parent child_node. child = cast child_node ⟹ h ⊢ get_parent child_node →r Some parent ⟹ P parent) ⟹ (⋀host shadow_root. child = cast shadow_root ⟹ h ⊢ get_host shadow_root →r host ⟹ P (cast host)) ⟹ P child) ⟹ P ptr"

lemma *l_get_parent_get_host_wf_is_get_parent_get_host_wf [instances]:*
"l_get_parent_get_host_wf heap_is_wellformed get_parent get_shadow_root get_host get_child_nodes"
⟨*proof*⟩

get_host

locale *l_get_host_wf*$_{Shadow_DOM}$ *=*
l_heap_is_wellformed$_{Shadow_DOM}$ *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 +*
l_type_wf type_wf +
l_get_host$_{Shadow_DOM}$ *get_shadow_root get_shadow_root_locs get_host get_host_locs type_wf +*
l_get_shadow_root type_wf get_shadow_root get_shadow_root_locs
for *known_ptr :: "(_::linorder) object_ptr ⇒ bool"*
and *known_ptrs :: "(_) heap ⇒ bool"*
and *type_wf :: "(_) heap ⇒ bool"*
and *get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"*
and *get_host_locs :: "((_) 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"
    and get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) 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 get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
begin

lemma get_host_ok [simp]:
  assumes "heap_is_wellformed h"
  assumes "type_wf h"
  assumes "known_ptrs h"
  assumes "shadow_root_ptr |∈| shadow_root_ptr_kinds h"
  shows "h ⊢ ok (get_host shadow_root_ptr)"
⟨proof⟩
end

interpretation i_get_host_wf?: l_get_host_wf_Shadow_DOM get_disconnected_document
  get_disconnected_document_locs known_ptr known_ptrs type_wf get_host get_host_locs 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 heap_is_wellformed parent_child_rel
  heap_is_wellformed_Core_DOM
  ⟨proof⟩
declare l_get_host_wf_Shadow_DOM_axioms [instances]

locale l_get_host_wf = l_heap_is_wellformed_defs + l_known_ptrs + l_type_wf + l_get_host_defs +
  assumes get_host_ok:
    "heap_is_wellformed h ⟹ known_ptrs h ⟹ type_wf h ⟹
    shadow_root_ptr |∈| shadow_root_ptr_kinds h ⟹ h ⊢ ok (get_host shadow_root_ptr)"

lemma get_host_wf_is_l_get_host_wf [instances]:
  "l_get_host_wf heap_is_wellformed known_ptr known_ptrs type_wf get_host"
  ⟨proof⟩
```

get_root_node_si

```
locale l_get_root_node_si_wf_Shadow_DOM =
  l_get_root_node_si_Shadow_DOM +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_parent_wf +
  l_get_parent_get_host_wf +
  l_get_host_wf
begin
lemma get_root_node_si_ptr_in_heap:
  assumes "h ⊢ ok (get_root_node_si ptr)"
  shows "ptr |∈| object_ptr_kinds h"
  ⟨proof⟩


lemma get_ancestors_si_ok:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
    and "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_ancestors_si ptr)"
⟨proof⟩

lemma get_ancestors_si_remains_not_in_ancestors:
  assumes "heap_is_wellformed h"
    and "heap_is_wellformed h'"
```

```
    and "h ⊢ get_ancestors_si ptr →r ancestors"
    and "h' ⊢ get_ancestors_si ptr →r ancestors'"
    and "⋀p children children'. h ⊢ get_child_nodes p →r children
        ⟹ h' ⊢ get_child_nodes p →r children' ⟹ set children' ⊆ set children"
    and "⋀p shadow_root_option shadow_root_option'. h ⊢ get_shadow_root p →r shadow_root_option ⟹
        h' ⊢ get_shadow_root p →r shadow_root_option' ⟹ (if shadow_root_option = None
        then shadow_root_option' = None else shadow_root_option' = None ∨
          shadow_root_option' = shadow_root_option)"
    and "node ∉ set ancestors"
    and object_ptr_kinds_eq3: "object_ptr_kinds h = object_ptr_kinds h'"
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
    and type_wf': "type_wf h'"
  shows "node ∉ set ancestors'"
⟨proof⟩


lemma get_ancestors_si_ptrs_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_ancestors_si ptr →r ancestors"
  assumes "ptr' ∈ set ancestors"
  shows "ptr' |∈| object_ptr_kinds h"
⟨proof⟩

lemma get_ancestors_si_reads:
  assumes "heap_is_wellformed h"
  shows "reads get_ancestors_si_locs (get_ancestors_si node_ptr) h h'"
⟨proof⟩


lemma get_ancestors_si_subset:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_si ptr →r ancestors"
    and "ancestor ∈ set ancestors"
    and "h ⊢ get_ancestors_si ancestor →r ancestor_ancestors"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "set ancestor_ancestors ⊆ set ancestors"
⟨proof⟩

lemma get_ancestors_si_also_parent:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_si some_ptr →r ancestors"
    and "cast child ∈ set ancestors"
    and "h ⊢ get_parent child →r Some parent"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "parent ∈ set ancestors"
⟨proof⟩

lemma get_ancestors_si_also_host:
  assumes "heap_is_wellformed h"
    and "h ⊢ get_ancestors_si some_ptr →r ancestors"
    and "cast shadow_root ∈ set ancestors"
    and "h ⊢ get_host shadow_root →r host"
    and type_wf: "type_wf h"
    and known_ptrs: "known_ptrs h"
  shows "cast host ∈ set ancestors"
⟨proof⟩

lemma get_ancestors_si_parent_child_rel:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
  assumes "h ⊢ get_ancestors_si child →r ancestors"
```

assumes *"((ptr, child) ∈ (parent_child_rel h)*)"*
shows *"ptr ∈ set ancestors"*
⟨*proof*⟩

lemma *get_ancestors_si_parent_child_host_shadow_root_rel:*
assumes *"heap_is_wellformed h"* **and** *"known_ptrs h"* **and** *"type_wf h"*
assumes *"h ⊢ get_ancestors_si child →r ancestors"*
assumes *"((ptr, child) ∈ (parent_child_rel h ∪ a_host_shadow_root_rel h)*)"*
shows *"ptr ∈ set ancestors"*
⟨*proof*⟩

lemma *get_root_node_si_ok:*
assumes *"heap_is_wellformed h"* **and** *"known_ptrs h"* **and** *"type_wf h"*
and *"ptr |∈| object_ptr_kinds h"*
shows *"h ⊢ ok (get_root_node_si ptr)"*
⟨*proof*⟩

lemma *get_root_node_si_root_in_heap:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ get_root_node_si ptr →r root"*
shows *"root |∈| object_ptr_kinds h"*
⟨*proof*⟩

lemma *get_root_node_si_same_no_parent:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ get_root_node_si ptr →r cast child"*
shows *"h ⊢ get_parent child →r None"*
⟨*proof*⟩
end

interpretation *i_get_root_node_si_wf?: l_get_root_node_si_wf*$_{Shadow_DOM}$ *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
⟨*proof*⟩
declare *l_get_root_node_si_wf*$_{Shadow_DOM}$*_axioms [instances]*

get_disconnected_document

locale *l_get_disconnected_document_wf*$_{Shadow_DOM}$ =
l_heap_is_wellformed$_{Shadow_DOM}$ +
l_get_disconnected_document$_{Core_DOM}$ +
l_get_parent_wf +
l_get_parent
begin

lemma *get_disconnected_document_ok:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ get_parent node_ptr →r None"*
shows *"h ⊢ ok (get_disconnected_document node_ptr)"*
⟨*proof*⟩
end

interpretation *i_get_disconnected_document_wf?: l_get_disconnected_document_wf*$_{Shadow_DOM}$
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 get_parent get_parent_locs
⟨*proof*⟩
declare *l_get_disconnected_document_wf*$_{Shadow_DOM}$*_axioms [instances]*

get_owner_document

```
locale l_get_owner_document_wf_Shadow_DOM =
  l_get_disconnected_nodes +
  l_get_child_nodes +
  l_get_owner_document_Shadow_DOM +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_parent_wf +
  l_known_ptrs +
  l_get_root_node_si_wf_Shadow_DOM +
  l_get_parent_Core_DOM +
  assumes known_ptr_impl: "known_ptr = ShadowRootClass.known_ptr"
begin
lemma get_owner_document_disconnected_nodes:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ get_disconnected_nodes document_ptr →_r disc_nodes"
  assumes "node_ptr ∈ set disc_nodes"
  assumes known_ptrs: "known_ptrs h"
  assumes type_wf: "type_wf h"
  shows "h ⊢ get_owner_document (cast node_ptr) →_r document_ptr"
⟨proof⟩

lemma in_disconnected_nodes_no_parent:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ get_parent node_ptr →_r None"
  assumes "h ⊢ get_owner_document (cast node_ptr) →_r owner_document"
  assumes "h ⊢ get_disconnected_nodes owner_document →_r disc_nodes"
  assumes "known_ptrs h"
  assumes "type_wf h"
  shows "node_ptr ∈ set disc_nodes"
⟨proof⟩


lemma get_owner_document_owner_document_in_heap_node:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ CD.a_get_owner_document_node_ptr node_ptr () →_r owner_document"
  shows "owner_document |∈| document_ptr_kinds h"
⟨proof⟩

lemma get_owner_document_owner_document_in_heap:
  assumes "heap_is_wellformed h" and "type_wf h" and "known_ptrs h"
  assumes "h ⊢ get_owner_document ptr →_r owner_document"
  shows "owner_document |∈| document_ptr_kinds h"
  ⟨proof⟩

lemma get_owner_document_ok:
  assumes "heap_is_wellformed h"  "known_ptrs h" "type_wf h"
  assumes "ptr |∈| object_ptr_kinds h"
  shows "h ⊢ ok (get_owner_document ptr)"
⟨proof⟩
end
interpretation i_get_owner_document_wf?: l_get_owner_document_wf_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
  ⟨proof⟩
declare l_get_owner_document_wf_Shadow_DOM_axioms [instances]


lemma get_owner_document_wf_is_l_get_owner_document_wf [instances]: "l_get_owner_document_wf
```

heap_is_wellformed type_wf known_ptr known_ptrs get_disconnected_nodes get_owner_document get_parent"
⟨*proof*⟩

remove_child

locale *l_remove_child_wf2*$_{Shadow_DOM}$ =
l_set_disconnected_nodes_get_disconnected_nodes +
l_get_child_nodes +
l_heap_is_wellformed$_{Shadow_DOM}$ *+*
l_get_owner_document_wf$_{Shadow_DOM}$ *+*
l_remove_child$_{Core_DOM}$ *+*
l_set_child_nodes_get_shadow_root +
l_set_disconnected_nodes_get_shadow_root +
l_set_child_nodes_get_tag_name +
l_set_disconnected_nodes_get_tag_name +
CD: l_remove_child_wf2$_{Core_DOM}$ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ *heap_is_wellformed*$_{Core_DOM}$
begin
lemma *remove_child_preserves_type_wf:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ remove_child ptr child →*$_h$ *h'"*
shows *"type_wf h'"*
⟨*proof*⟩

lemma *remove_child_preserves_known_ptrs:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ remove_child ptr child →*$_h$ *h'"*
shows *"known_ptrs h'"*
⟨*proof*⟩

lemma *remove_child_heap_is_wellformed_preserved:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ remove_child ptr child →*$_h$ *h'"*
shows *"heap_is_wellformed h'"*
⟨*proof*⟩

lemma *remove_preserves_type_wf:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ remove child →*$_h$ *h'"*
shows *"type_wf h'"*
⟨*proof*⟩

lemma *remove_preserves_known_ptrs:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ remove child →*$_h$ *h'"*
shows *"known_ptrs h'"*
⟨*proof*⟩

lemma *remove_heap_is_wellformed_preserved:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ remove child →*$_h$ *h'"*
shows *"heap_is_wellformed h'"*
⟨*proof*⟩

lemma *remove_child_removes_child:*
"heap_is_wellformed h ⟹ h ⊢ remove_child ptr' child →$_h$ *h'*
⟹ h' ⊢ get_child_nodes ptr →$_r$ *children*
⟹ known_ptrs h ⟹ type_wf h
⟹ child ∉ set children"
⟨*proof*⟩
lemma *remove_child_removes_first_child: "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹*

```
    h ⊢ get_child_nodes ptr →r node_ptr # children ⟹ h ⊢ remove_child ptr node_ptr →h h' ⟹
    h' ⊢ get_child_nodes ptr →r children"
  ⟨proof⟩
lemma remove_removes_child: "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹
    h ⊢ get_child_nodes ptr →r node_ptr # children ⟹ h ⊢ remove node_ptr →h h' ⟹
    h' ⊢ get_child_nodes ptr →r children"
  ⟨proof⟩
lemma remove_for_all_empty_children: "heap_is_wellformed h ⟹ type_wf h ⟹ known_ptrs h ⟹
    h ⊢ get_child_nodes ptr →r children ⟹ h ⊢ forall_M remove children →h h' ⟹
    h' ⊢ get_child_nodes ptr →r []"
  ⟨proof⟩
end

interpretation i_remove_child_wf2?: l_remove_child_wf2_Shadow_DOM
  type_wf get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes
  set_disconnected_nodes_locs known_ptr get_child_nodes get_child_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_host get_host_locs get_disconnected_document
  get_disconnected_document_locs DocumentClass.known_ptr get_parent get_parent_locs get_root_node_si
  get_root_node_si_locs CD.a_get_owner_document get_owner_document known_ptrs get_ancestors_si
  get_ancestors_si_locs set_child_nodes set_child_nodes_locs remove_child remove_child_locs remove
  ⟨proof⟩
declare l_remove_child_wf2_Shadow_DOM_axioms [instances]

lemma remove_child_wf2_is_l_remove_child_wf2 [instances]:
  "l_remove_child_wf2 type_wf known_ptr known_ptrs remove_child heap_is_wellformed get_child_nodes remove"
  ⟨proof⟩
```

adopt_node

```
locale l_adopt_node_wf2_Shadow_DOM =
  l_get_child_nodes +
  l_get_disconnected_nodes +
  l_set_child_nodes_get_shadow_root +
  l_set_disconnected_nodes_get_shadow_root +
  l_set_child_nodes_get_tag_name +
  l_set_disconnected_nodes_get_tag_name +
  l_heap_is_wellformed_Shadow_DOM +
  l_get_root_node +
  l_set_disconnected_nodes_get_child_nodes +
  l_get_owner_document_wf +
  l_remove_child_wf2 +
  l_adopt_node_wf_Core_DOM +
  l_adopt_node_Core_DOM +
  l_get_parent_wf_Core_DOM
begin

lemma adopt_node_removes_child:
  assumes wellformed: "heap_is_wellformed h"
    and adopt_node: "h ⊢ adopt_node owner_document node_ptr →h h2"
    and children: "h2 ⊢ get_child_nodes ptr →r children"
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
  shows "node_ptr ∉ set children"
⟨proof⟩

lemma adopt_node_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ adopt_node document_ptr child →h h'"
    and known_ptrs: "known_ptrs h"
    and type_wf: "type_wf h"
  shows "heap_is_wellformed h'" and "known_ptrs h'" and "type_wf h'"
⟨proof⟩
```

lemma *adopt_node_node_in_disconnected_nodes:*
assumes *wellformed: "heap_is_wellformed h"*
and *adopt_node: "h ⊢ adopt_node owner_document node_ptr →$_h$ h'"*
and *"h' ⊢ get_disconnected_nodes owner_document →$_r$ disc_nodes"*
and *known_ptrs: "known_ptrs h"*
and *type_wf: "type_wf h"*
shows *"node_ptr ∈ set disc_nodes"*
⟨*proof*⟩
end

interpretation *l_adopt_node_wf$_{Core_DOM}$ Shadow_DOM.get_owner_document Shadow_DOM.get_parent*
Shadow_DOM.get_parent_locs Shadow_DOM.remove_child Shadow_DOM.remove_child_locs
get_disconnected_nodes get_disconnected_nodes_locs
set_disconnected_nodes set_disconnected_nodes_locs Shadow_DOM.adopt_node Shadow_DOM.adopt_node_locs
ShadowRootClass.known_ptr ShadowRootClass.type_wf Shadow_DOM.get_child_nodes
Shadow_DOM.get_child_nodes_locs
ShadowRootClass.known_ptrs Shadow_DOM.set_child_nodes Shadow_DOM.set_child_nodes_locs
Shadow_DOM.remove Shadow_DOM.heap_is_wellformed Shadow_DOM.parent_child_rel
⟨*proof*⟩
declare *l_adopt_node_wf$_{Core_DOM}$_axioms [instances]*

interpretation *i_adopt_node_wf2?: l_adopt_node_wf2$_{Shadow_DOM}$*
type_wf known_ptr get_child_nodes get_child_nodes_locs get_disconnected_nodes
get_disconnected_nodes_locs set_child_nodes set_child_nodes_locs get_shadow_root get_shadow_root_locs
set_disconnected_nodes set_disconnected_nodes_locs get_tag_name get_tag_name_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_root_node get_root_node_locs get_parent get_parent_locs known_ptrs
get_owner_document remove_child remove_child_locs remove adopt_node adopt_node_locs
⟨*proof*⟩
declare *l_adopt_node_wf2$_{Shadow_DOM}$_axioms [instances]*

lemma *adopt_node_wf_is_l_adopt_node_wf [instances]:*
"l_adopt_node_wf type_wf known_ptr heap_is_wellformed parent_child_rel get_child_nodes
get_disconnected_nodes known_ptrs adopt_node"
⟨*proof*⟩

insert_before

locale *l_insert_before_wf2$_{Shadow_DOM}$ =*
l_get_child_nodes +
l_get_disconnected_nodes +
l_set_child_nodes_get_shadow_root +
l_set_disconnected_nodes_get_shadow_root +
l_set_child_nodes_get_tag_name +
l_set_disconnected_nodes_get_tag_name +
l_set_disconnected_nodes_get_disconnected_nodes +
l_set_child_nodes_get_disconnected_nodes +
l_set_disconnected_nodes_get_disconnected_nodes_wf +
l_set_disconnected_nodes_get_ancestors_si +
l_insert_before$_{Core_DOM}$ _ _ _ _ _ _ get_ancestors_si get_ancestors_si_locs +
l_get_root_node_si_wf$_{Shadow_DOM}$ +
l_get_owner_document +
l_adopt_node +
l_adopt_node_wf +
l_heap_is_wellformed$_{Shadow_DOM}$ +
l_adopt_node_get_shadow_root
begin
lemma *insert_before_child_preserves:*
assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ insert_before ptr node child →$_h$ h'"*

shows *"type_wf h'"* **and** *"known_ptrs h'"* **and** *"heap_is_wellformed h'"*
⟨*proof*⟩
end

interpretation *i_insert_before_wf?: l_insert_before_wf*$_{Core_DOM}$ *get_parent get_parent_locs*
get_child_nodes get_child_nodes_locs set_child_nodes set_child_nodes_locs get_ancestors_si
get_ancestors_si_locs adopt_node adopt_node_locs set_disconnected_nodes set_disconnected_nodes_locs
get_disconnected_nodes get_disconnected_nodes_locs get_owner_document insert_before
insert_before_locs append_child type_wf known_ptr known_ptrs heap_is_wellformed parent_child_rel
⟨*proof*⟩
declare *l_insert_before_wf*$_{Core_DOM}$*_axioms [instances]*

lemma *insert_before_wf_is_l_insert_before_wf [instances]:*
"l_insert_before_wf Shadow_DOM.heap_is_wellformed ShadowRootClass.type_wf
ShadowRootClass.known_ptr ShadowRootClass.known_ptrs
Shadow_DOM.insert_before Shadow_DOM.get_child_nodes"
⟨*proof*⟩

lemma *l_set_disconnected_nodes_get_disconnected_nodes_wf [instances]: "l_set_disconnected_nodes_get_disconnected_*
ShadowRootClass.type_wf
ShadowRootClass.known_ptr Shadow_DOM.heap_is_wellformed Shadow_DOM.parent_child_rel Shadow_DOM.get_child_nodes
get_disconnected_nodes get_disconnected_nodes_locs set_disconnected_nodes set_disconnected_nodes_locs"
⟨*proof*⟩

interpretation *l_get_root_node_si_wf*$_{Shadow_DOM}$
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
⟨*proof*⟩
declare *l_get_root_node_si_wf*$_{Shadow_DOM}$*_axioms [instances]*

interpretation *i_insert_before_wf2?: l_insert_before_wf2*$_{Shadow_DOM}$ *type_wf known_ptr get_child_nodes*
get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs set_child_nodes set_child_nodes_locs
get_shadow_root get_shadow_root_locs set_disconnected_nodes set_disconnected_nodes_locs get_tag_name get_tag_name_l
heap_is_wellformed parent_child_rel get_ancestors_si get_ancestors_si_locs get_parent get_parent_locs adopt_node
adopt_node_locs get_owner_document insert_before insert_before_locs append_child known_ptrs get_host get_host_locs
get_root_node_si get_root_node_si_locs heap_is_wellformed$_{Core_DOM}$ *get_disconnected_document get_disconnected_docu*
⟨*proof*⟩
declare *l_insert_before_wf2*$_{Shadow_DOM}$*_axioms [instances]*

lemma *insert_before_wf2_is_l_insert_before_wf2 [instances]:*
"l_insert_before_wf2 ShadowRootClass.type_wf ShadowRootClass.known_ptr ShadowRootClass.known_ptrs
Shadow_DOM.insert_before
Shadow_DOM.heap_is_wellformed"
⟨*proof*⟩

append_child

interpretation *i_append_child_wf?: l_append_child_wf*$_{Core_DOM}$ *get_owner_document get_parent*
get_parent_locs remove_child remove_child_locs
get_disconnected_nodes get_disconnected_nodes_locs
set_disconnected_nodes set_disconnected_nodes_locs
adopt_node adopt_node_locs known_ptr type_wf get_child_nodes
get_child_nodes_locs known_ptrs set_child_nodes
set_child_nodes_locs remove get_ancestors_si get_ancestors_si_locs
insert_before insert_before_locs append_child heap_is_wellformed
parent_child_rel
⟨*proof*⟩
declare *l_append_child_wf*$_{Core_DOM}$*_axioms [instances]*

lemma *append_child_wf_is_l_append_child_wf [instances]:*
"l_append_child_wf type_wf known_ptr known_ptrs append_child heap_is_wellformed"
⟨*proof*⟩

to_tree_order

interpretation *i_to_tree_order_wf?: l_to_tree_order_wf*$_{Core_DOM}$ *known_ptr type_wf get_child_nodes get_child_nodes_locs to_tree_order known_ptrs get_parent get_parent_locs heap_is_wellformed parent_child_rel get_disconnected_nodes get_disconnected_nodes_locs*
⟨*proof*⟩
declare *l_to_tree_order_wf*$_{Core_DOM}$*_axioms [instances]*

lemma *to_tree_order_wf_is_l_to_tree_order_wf [instances]:*
"l_to_tree_order_wf heap_is_wellformed parent_child_rel type_wf known_ptr known_ptrs to_tree_order get_parent get_child_nodes"
⟨*proof*⟩

get_root_node

interpretation *i_to_tree_order_wf_get_root_node_wf?: l_to_tree_order_wf_get_root_node_wf*$_{Core_DOM}$ *known_ptr type_wf known_ptrs heap_is_wellformed parent_child_rel get_child_nodes get_child_nodes_locs get_disconnected_nodes get_disconnected_nodes_locs get_parent get_parent_locs get_ancestors get_ancestors_locs get_root_node get_root_node_locs to_tree_order*
⟨*proof*⟩
declare *l_to_tree_order_wf_get_root_node_wf*$_{Core_DOM}$*_axioms [instances]*

lemma *to_tree_order_wf_get_root_node_wf_is_l_to_tree_order_wf_get_root_node_wf [instances]:*
"l_to_tree_order_wf_get_root_node_wf ShadowRootClass.type_wf ShadowRootClass.known_ptr ShadowRootClass.known_ptrs to_tree_order Shadow_DOM.get_root_node Shadow_DOM.heap_is_wellformed"
⟨*proof*⟩

to_tree_order_si

locale *l_to_tree_order_si_wf*$_{Shadow_DOM}$ =
l_get_child_nodes +
l_get_parent_get_host_wf$_{Shadow_DOM}$ +
l_to_tree_order_si$_{Shadow_DOM}$
begin
lemma *to_tree_order_si_ok:*
assumes *"heap_is_wellformed h"* **and** *"known_ptrs h"* **and** *"type_wf h"*
and *"ptr |∈| object_ptr_kinds h"*
shows *"h ⊢ ok (to_tree_order_si ptr)"*
⟨*proof*⟩
end

interpretation *i_to_tree_order_si_wf?: l_to_tree_order_si_wf*$_{Shadow_DOM}$ *type_wf known_ptr 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 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 get_parent get_parent_locs to_tree_order_si*
⟨*proof*⟩
declare *l_to_tree_order_si_wf*$_{Shadow_DOM}$*_axioms [instances]*

get_assigned_nodes

lemma *forall_M_small_big: "h ⊢ forall_M f xs →*$_h$ *h' ⟹ P h ⟹ (⋀h h' x. x ∈ set xs ⟹ h ⊢ f x →*$_h$ *h' ⟹ P h ⟹ P h') ⟹ P h'"*
⟨*proof*⟩

locale *l_assigned_nodes_wf*$_{Shadow_DOM}$ =
l_assigned_nodes$_{Shadow_DOM}$ +
l_heap_is_wellformed +

```
  l_remove_child_wf2 +
  l_append_child_wf +
  l_remove_shadow_root_wf_Shadow_DOM
begin

lemma assigned_nodes_distinct:
  assumes "heap_is_wellformed h"
  assumes "h ⊢ assigned_nodes slot →_r nodes"
  shows "distinct nodes"
⟨proof⟩


lemma flatten_dom_preserves:
  assumes "heap_is_wellformed h" and "known_ptrs h" and "type_wf h"
  assumes "h ⊢ flatten_dom →_h h'"
  shows "heap_is_wellformed h'" and "known_ptrs h'" and "type_wf h'"
⟨proof⟩
end

interpretation i_assigned_nodes_wf?: l_assigned_nodes_wf_Shadow_DOM
  known_ptr assigned_nodes assigned_nodes_flatten flatten_dom get_child_nodes get_child_nodes_locs
  get_tag_name get_tag_name_locs get_root_node get_root_node_locs get_host get_host_locs find_slot
  assigned_slot remove insert_before insert_before_locs append_child remove_shadow_root
  remove_shadow_root_locs type_wf get_shadow_root get_shadow_root_locs set_shadow_root
  set_shadow_root_locs get_parent get_parent_locs to_tree_order heap_is_wellformed parent_child_rel
  get_disconnected_nodes get_disconnected_nodes_locs known_ptrs remove_child remove_child_locs
  heap_is_wellformed_Core_DOM get_disconnected_document get_disconnected_document_locs
  ⟨proof⟩
declare l_assigned_nodes_wf_Shadow_DOM_axioms [instances]
```

get_shadow_root_safe

```
locale l_get_shadow_root_safe_wf_Shadow_DOM =
  l_heap_is_wellformed_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 +
  l_type_wf type_wf +
  l_get_shadow_root_safe_Shadow_DOM type_wf get_shadow_root_safe get_shadow_root_safe_locs
  get_shadow_root get_shadow_root_locs get_mode get_mode_locs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and known_ptrs :: "(_) heap ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) 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"
    and get_shadow_root_safe ::
    "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
    and get_shadow_root_safe_locs ::
    "(_) element_ptr ⇒ (_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_child_nodes :: "(_::linorder) object_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_child_nodes_locs :: "(_) object_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) 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 get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
    and get_mode :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, shadow_root_mode) prog"
    and get_mode_locs :: "(_) shadow_root_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
```

begin

end

create_element

locale *l_create_element_wf*$_{Shadow_DOM}$ =
l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs +
l_set_tag_name type_wf set_tag_name set_tag_name_locs +
l_create_element_defs create_element +
l_heap_is_wellformed$_{Shadow_DOM}$ *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 +
l_new_element_get_disconnected_nodes get_disconnected_nodes get_disconnected_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_create_element$_{Shadow_DOM}$ *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
type_wf$_{Core_DOM}$ *known_ptr*$_{Core_DOM}$ *+*
l_new_element_get_child_nodes type_wf known_ptr get_child_nodes get_child_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_tag_name type_wf get_tag_name get_tag_name_locs set_tag_name set_tag_name_locs +
l_new_element_get_tag_name type_wf get_tag_name get_tag_name_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_shadow_root set_disconnected_nodes set_disconnected_nodes_locs
get_shadow_root get_shadow_root_locs +
l_set_disconnected_nodes_get_tag_name type_wf set_disconnected_nodes set_disconnected_nodes_locs
get_tag_name get_tag_name_locs +
l_set_disconnected_nodes type_wf set_disconnected_nodes set_disconnected_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_get_shadow_root type_wf get_shadow_root get_shadow_root_locs +
l_set_tag_name_get_shadow_root type_wf set_tag_name set_tag_name_locs get_shadow_root
get_shadow_root_locs +
l_new_element type_wf +
l_known_ptrs known_ptr known_ptrs
for *known_ptr :: "(_::linorder) object_ptr ⇒ bool"*
and *known_ptrs :: "(_) heap ⇒ bool"*
and *type_wf :: "(_) heap ⇒ bool"*
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_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"*
and *get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
and *heap_is_wellformed :: "(_) heap ⇒ bool"*
and *parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) 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 *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_element ::*
"(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
and *get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"*
and *get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
and *get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"*
and *get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"*
and *heap_is_wellformed*$_{Core_DOM}$ *:: "(_) heap ⇒ bool"*

```
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
begin
lemma create_element_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ create_element document_ptr tag →_h h'"
    and "type_wf h"
    and "known_ptrs h"
  shows "heap_is_wellformed h'" and "type_wf h'" and "known_ptrs h'"
⟨proof⟩
end

interpretation i_create_element_wf?: l_create_element_wf_Shadow_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 get_shadow_root get_shadow_root_locs get_tag_name
  get_tag_name_locs heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs DocumentClass.known_ptr DocumentClass.type_wf
  ⟨proof⟩
declare l_create_element_wf_Core_DOM_axioms [instances]
```

create_character_data

```
locale l_create_character_data_wf_Shadow_DOM =
  l_get_disconnected_nodes type_wf get_disconnected_nodes get_disconnected_nodes_locs +
  l_heap_is_wellformed_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 +
  l_create_character_data_Shadow_DOM get_disconnected_nodes get_disconnected_nodes_locs
  set_disconnected_nodes set_disconnected_nodes_locs set_val set_val_locs create_character_data
  known_ptr type_wf_Core_DOM known_ptr_Core_DOM
  + l_new_character_data_get_disconnected_nodes
  get_disconnected_nodes get_disconnected_nodes_locs

+ l_set_val_get_disconnected_nodes
type_wf set_val set_val_locs get_disconnected_nodes get_disconnected_nodes_locs
+ l_new_character_data_get_child_nodes
type_wf known_ptr get_child_nodes get_child_nodes_locs
+ l_set_val_get_child_nodes
type_wf set_val set_val_locs known_ptr get_child_nodes get_child_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
type_wf set_disconnected_nodes set_disconnected_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_set_val_get_shadow_root type_wf set_val set_val_locs get_shadow_root get_shadow_root_locs
+ l_set_disconnected_nodes_get_shadow_root set_disconnected_nodes set_disconnected_nodes_locs
get_shadow_root get_shadow_root_locs
+ l_new_character_data_get_tag_name
get_tag_name get_tag_name_locs
+ l_set_val_get_tag_name type_wf set_val set_val_locs get_tag_name get_tag_name_locs
+ l_get_tag_name type_wf get_tag_name get_tag_name_locs
+ l_set_disconnected_nodes_get_tag_name type_wf set_disconnected_nodes set_disconnected_nodes_locs
get_tag_name get_tag_name_locs
```

```
+ l_new_character_data
type_wf
+ l_known_ptrs
known_ptr known_ptrs
for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
  and known_ptrs :: "(_) heap ⇒ bool"
  and type_wf :: "(_) heap ⇒ bool"
  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_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
  and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed :: "(_) heap ⇒ bool"
  and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) 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 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_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
  and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
  and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
  and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
  and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  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 create_character_data ::
  "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) character_data_ptr) prog"
  and known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
  and type_wf_Core_DOM :: "(_) heap ⇒ bool"
begin
lemma create_character_data_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ create_character_data document_ptr text →_h h'"
    and "type_wf h"
    and "known_ptrs h"
  shows "heap_is_wellformed h'" and "type_wf h'" and "known_ptrs h'"
⟨proof⟩
end
```

create_document

```
locale l_create_document_wf_Shadow_DOM =
  l_heap_is_wellformed_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
  + l_new_document_get_disconnected_nodes
  get_disconnected_nodes get_disconnected_nodes_locs
  + l_create_document_Core_DOM
  create_document
  + l_new_document_get_child_nodes
  type_wf known_ptr get_child_nodes get_child_nodes_locs
  + l_get_tag_name type_wf get_tag_name get_tag_name_locs
  + l_new_document_get_tag_name get_tag_name get_tag_name_locs
  + l_get_disconnected_nodes_Shadow_DOM type_wf type_wf_Core_DOM get_disconnected_nodes
  get_disconnected_nodes_locs
  + l_new_document
```

```
  type_wf
  + l_known_ptrs
  known_ptr known_ptrs
  for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
    and type_wf :: "(_) heap ⇒ bool"
    and type_wf_Core_DOM :: "(_) heap ⇒ bool"
    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_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
    and get_disconnected_nodes_locs :: "(_) document_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"
    and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
    and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
    and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
    and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
    and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
    and heap_is_wellformed :: "(_) heap ⇒ bool"
    and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) set"
    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 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_document :: "((_) heap, exception, (_) document_ptr) prog"
    and known_ptrs :: "(_) heap ⇒ bool"
begin

lemma create_document_preserves_wellformedness:
  assumes "heap_is_wellformed h"
    and "h ⊢ create_document →_h h'"
    and "type_wf h"
    and "known_ptrs h"
  shows "heap_is_wellformed h'"
⟨proof⟩
end

interpretation l_create_document_wf_Shadow_DOM known_ptr type_wf DocumentClass.type_wf 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
  heap_is_wellformed_Core_DOM get_host get_host_locs get_disconnected_document
  get_disconnected_document_locs heap_is_wellformed parent_child_rel set_val set_val_locs
  set_disconnected_nodes set_disconnected_nodes_locs create_document known_ptrs
  ⟨proof⟩
```

attach_shadow_root

```
locale l_attach_shadow_root_wf_Shadow_DOM =
  l_get_disconnected_nodes
  type_wf get_disconnected_nodes get_disconnected_nodes_locs
  + l_heap_is_wellformed_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
  + 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_new_shadow_root_get_disconnected_nodes
  get_disconnected_nodes get_disconnected_nodes_locs
```

```
+ l_set_mode_get_disconnected_nodes
type_wf set_mode set_mode_locs get_disconnected_nodes get_disconnected_nodes_locs
+ l_new_shadow_root_get_child_nodes
type_wf known_ptr get_child_nodes get_child_nodes_locs
+ l_new_shadow_root_get_tag_name
type_wf get_tag_name get_tag_name_locs
+ l_set_mode_get_child_nodes
type_wf set_mode set_mode_locs known_ptr get_child_nodes get_child_nodes_locs
+ l_set_shadow_root_get_child_nodes
type_wf set_shadow_root set_shadow_root_locs known_ptr get_child_nodes get_child_nodes_locs
+ l_set_shadow_root
type_wf set_shadow_root set_shadow_root_locs
+ l_set_shadow_root_get_disconnected_nodes
set_shadow_root set_shadow_root_locs get_disconnected_nodes get_disconnected_nodes_locs
+ l_set_mode_get_shadow_root type_wf set_mode set_mode_locs get_shadow_root get_shadow_root_locs
+ l_set_shadow_root_get_shadow_root type_wf set_shadow_root set_shadow_root_locs
get_shadow_root get_shadow_root_locs
+ l_new_character_data_get_tag_name
get_tag_name get_tag_name_locs
+ l_set_mode_get_tag_name type_wf set_mode set_mode_locs get_tag_name get_tag_name_locs
+ l_get_tag_name type_wf get_tag_name get_tag_name_locs
+ l_set_shadow_root_get_tag_name set_shadow_root set_shadow_root_locs get_tag_name get_tag_name_locs
+ l_new_shadow_root
type_wf
+ l_known_ptrs
known_ptr known_ptrs
for known_ptr :: "(_::linorder) object_ptr ⇒ bool"
  and known_ptrs :: "(_) heap ⇒ bool"
  and type_wf :: "(_) heap ⇒ bool"
  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_disconnected_nodes :: "(_) document_ptr ⇒ ((_) heap, exception, (_) node_ptr list) prog"
  and get_disconnected_nodes_locs :: "(_) document_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed :: "(_) heap ⇒ bool"
  and parent_child_rel :: "(_) heap ⇒ ((_) object_ptr × (_) object_ptr) 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 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_element :: "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_shadow_root :: "(_) element_ptr ⇒ ((_) heap, exception, (_) shadow_root_ptr option) prog"
  and get_shadow_root_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_tag_name :: "(_) element_ptr ⇒ ((_) heap, exception, char list) prog"
  and get_tag_name_locs :: "(_) element_ptr ⇒ ((_) heap ⇒ (_) heap ⇒ bool) set"
  and heap_is_wellformed_Core_DOM :: "(_) heap ⇒ bool"
  and get_host :: "(_) shadow_root_ptr ⇒ ((_) heap, exception, (_) element_ptr) prog"
  and get_host_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  and get_disconnected_document :: "(_) node_ptr ⇒ ((_) heap, exception, (_) document_ptr) prog"
  and get_disconnected_document_locs :: "((_) heap ⇒ (_) heap ⇒ bool) set"
  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 create_character_data ::
  "(_) document_ptr ⇒ char list ⇒ ((_) heap, exception, (_) character_data_ptr) prog"
  and known_ptr_Core_DOM :: "(_::linorder) object_ptr ⇒ bool"
  and type_wf_Core_DOM :: "(_) heap ⇒ bool"
  and set_shadow_root :: "(_) element_ptr ⇒ (_) shadow_root_ptr option ⇒ (_, unit) dom_prog"
  and set_shadow_root_locs :: "(_) element_ptr ⇒ (_, unit) dom_prog set"
  and set_mode :: "(_) shadow_root_ptr ⇒ shadow_root_mode ⇒ (_, unit) dom_prog"
  and set_mode_locs :: "(_) shadow_root_ptr ⇒ (_, unit) dom_prog set"
  and attach_shadow_root :: "(_) element_ptr ⇒ shadow_root_mode ⇒ (_, (_) shadow_root_ptr) dom_prog"
begin
lemma attach_shadow_root_child_preserves:
```

assumes *"heap_is_wellformed h"* **and** *"type_wf h"* **and** *"known_ptrs h"*
assumes *"h ⊢ attach_shadow_root element_ptr new_mode →$_h$ h'"*
shows *"type_wf h'"* **and** *"known_ptrs h'"* **and** *"heap_is_wellformed h'"*
⟨*proof*⟩
end

interpretation *l_attach_shadow_root_wf?: l_attach_shadow_root_wf$_{Shadow_DOM}$ 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 get_shadow_root get_shadow_root_locs get_tag_name get_tag_name_locs heap_is_wellformed$_{Core_DOM}$ get_host get_host_locs get_disconnected_document get_disconnected_document_locs set_val set_val_locs create_character_data DocumentClass.known_ptr DocumentClass.type_wf set_shadow_root set_shadow_root_locs set_mode set_mode_locs attach_shadow_root*
⟨*proof*⟩
declare *l_attach_shadow_root_wf$_{Shadow_DOM}$_axioms [instances]*

end

3 Test Suite

In this chapter, we present the formalized compliance test cases for the core DOM. As our formalization is executable, we can (symbolically) execute the test cases on top of our model. Executing these test cases successfully shows that our model is compliant to the official DOM standard. As future work, we plan to generate test cases from our formal model (e.g., using [7, 9]) to improve the quality of the official compliance test suite. For more details on the relation of test and proof in the context of web standards, we refer the reader to [4].

3.1 Shadow DOM Base Tests (Shadow_DOM_BaseTest)

```
theory Shadow_DOM_BaseTest
  imports
    "../Shadow_DOM"
begin


definition "assert_throws e p = do {
  h ← get_heap;
  (if (h ⊢ p →_e e) then return () else error AssertException)
}"
notation assert_throws (<assert'_throws'(_, _')>)

definition "test p h ⟷ h ⊢ ok p"


definition field_access :: "(string ⇒ (_, (_) object_ptr option) dom_prog) ⇒ string ⇒
(_, (_) object_ptr option) dom_prog"  (infix <.> 80)
  where
    "field_access m field = m field"

definition assert_equals :: "'a ⇒ 'a ⇒ (_, unit) dom_prog"
  where
    "assert_equals l r = (if l = r then return () else error AssertException)"
definition assert_equals_with_message :: "'a ⇒ 'a ⇒ 'b ⇒ (_, unit) dom_prog"
  where
    "assert_equals_with_message l r _ = (if l = r then return () else error AssertException)"
notation assert_equals (<assert'_equals'(_, _')>)
notation assert_equals_with_message (<assert'_equals'(_, _, _')>)
notation assert_equals (<assert'_array'_equals'(_, _')>)
notation assert_equals_with_message (<assert'_array'_equals'(_, _, _')>)

definition assert_not_equals :: "'a ⇒ 'a ⇒ (_, unit) dom_prog"
  where
    "assert_not_equals l r = (if l ≠ r then return () else error AssertException)"
definition assert_not_equals_with_message :: "'a ⇒ 'a ⇒ 'b ⇒ (_, unit) dom_prog"
  where
    "assert_not_equals_with_message l r _ = (if l ≠ r then return () else error AssertException)"
notation assert_not_equals (<assert'_not'_equals'(_, _')>)
notation assert_not_equals_with_message (<assert'_not'_equals'(_, _, _')>)
notation assert_not_equals (<assert'_array'_not'_equals'(_, _')>)
notation assert_not_equals_with_message (<assert'_array'_not'_equals'(_, _, _')>)


definition removeWhiteSpaceOnlyTextNodes :: "((_) object_ptr option) ⇒ (_, unit) dom_prog"
  where
```

```
  "removeWhiteSpaceOnlyTextNodes _ = return ()"
```

3.1.1 Making the functions under test compatible with untyped languages such as JavaScript

```
fun set_attribute_with_null :: "((_) object_ptr option) ⇒ attr_key ⇒ attr_value ⇒ (_, unit) dom_prog"
  where
    "set_attribute_with_null (Some ptr) k v = (case cast ptr of
      Some element_ptr ⇒ set_attribute element_ptr k (Some v))"
fun set_attribute_with_null2 :: "((_) object_ptr option) ⇒ attr_key ⇒ attr_value option ⇒ (_, unit) dom_prog"
  where
    "set_attribute_with_null2 (Some ptr) k v = (case cast ptr of
      Some element_ptr ⇒ set_attribute element_ptr k v)"
notation set_attribute_with_null (<_ . setAttribute'(_, _')>)
notation set_attribute_with_null2 (<_ . setAttribute'(_, _')>)

fun get_child_nodes_Core_DOM_with_null :: "((_) object_ptr option) ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "get_child_nodes_Core_DOM_with_null (Some ptr) = do {
      children ← get_child_nodes ptr;
      return (map (Some ∘ cast) children)
    }"
notation get_child_nodes_Core_DOM_with_null (<_ . childNodes>)

fun create_element_with_null :: "((_) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "create_element_with_null (Some owner_document_obj) tag = (case cast owner_document_obj of
      Some owner_document ⇒ do {
        element_ptr ← create_element owner_document tag;
        return (Some (cast element_ptr))})"
notation create_element_with_null (<_ . createElement'(_')>)

fun create_character_data_with_null :: "((_) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option))
dom_prog"
  where
    "create_character_data_with_null (Some owner_document_obj) tag = (case cast owner_document_obj of
      Some owner_document ⇒ do {
        character_data_ptr ← create_character_data owner_document tag;
        return (Some (cast character_data_ptr))})"
notation create_character_data_with_null (<_ . createTextNode'(_')>)

definition create_document_with_null :: "string ⇒ (_, ((_::linorder) object_ptr option)) dom_prog"
  where
    "create_document_with_null title = do {
      new_document_ptr ← create_document;
      html ← create_element new_document_ptr ''html'';
      append_child (cast new_document_ptr) (cast html);
      heap ← create_element new_document_ptr ''heap'';
      append_child (cast html) (cast heap);
      body ← create_element new_document_ptr ''body'';
      append_child (cast html) (cast body);
      return (Some (cast new_document_ptr))
    }"
abbreviation "create_document_with_null2 _ _ _ ≡ create_document_with_null ''''"
notation create_document_with_null (<createDocument'(_')>)
notation create_document_with_null2 (<createDocument'(_, _, _')>)

fun get_element_by_id_with_null ::
  "((_::linorder) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_element_by_id_with_null (Some ptr) id' = do {
        element_ptr_opt ← get_element_by_id ptr id';
        (case element_ptr_opt of
```

```
        Some element_ptr ⇒ return (Some (cast_element_ptr2object_ptr element_ptr))
      | None ⇒ return None)}"
  | "get_element_by_id_with_null _ _ = error SegmentationFault"
notation get_element_by_id_with_null (<_ . getElementById'(_')>)

fun get_elements_by_class_name_with_null ::
  "((_::linorder) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option) list) dom_prog"
  where
    "get_elements_by_class_name_with_null (Some ptr) class_name =
      get_elements_by_class_name ptr class_name ≫= map_M (return ∘ Some ∘ cast_element_ptr2object_ptr)"
notation get_elements_by_class_name_with_null (<_ . getElementsByClassName'(_')>)

fun get_elements_by_tag_name_with_null ::
  "((_::linorder) object_ptr option) ⇒ string ⇒ (_, ((_) object_ptr option) list) dom_prog"
  where
    "get_elements_by_tag_name_with_null (Some ptr) tag =
      get_elements_by_tag_name ptr tag ≫= map_M (return ∘ Some ∘ cast_element_ptr2object_ptr)"
notation get_elements_by_tag_name_with_null (<_ . getElementsByTagName'(_')>)

fun insert_before_with_null ::
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ ((_) object_ptr option) ⇒
(_, ((_) object_ptr option)) dom_prog"
  where
    "insert_before_with_null (Some ptr) (Some child_obj) ref_child_obj_opt = (case cast child_obj of
      Some child ⇒ do {
        (case ref_child_obj_opt of
          Some ref_child_obj ⇒ insert_before ptr child (cast ref_child_obj)
        | None ⇒ insert_before ptr child None);
        return (Some child_obj)}
    | None ⇒ error HierarchyRequestError)"
notation insert_before_with_null (<_ . insertBefore'(_, _')>)

fun append_child_with_null ::
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ (_, unit) dom_prog"
  where
    "append_child_with_null (Some ptr) (Some child_obj) = (case cast child_obj of
      Some child ⇒ append_child ptr child
    | None ⇒ error SegmentationFault)"
notation append_child_with_null (<_ . appendChild'(_')>)
code_thms append_child_with_null
fun get_body :: "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_body ptr = do {
       ptrs ← ptr . getElementsByTagName(''body'');
       return (hd ptrs)
    }"
notation get_body (<_ . body>)

fun get_document_element_with_null ::
  "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_document_element_with_null (Some ptr) = (case cast_object_ptr2document_ptr ptr of
    Some document_ptr ⇒ do {
      element_ptr_opt ← get_M document_ptr document_element;
      return (case element_ptr_opt of
        Some element_ptr ⇒ Some (cast_element_ptr2object_ptr element_ptr)
      | None ⇒ None)})"
notation get_document_element_with_null (<_ . documentElement>)

fun get_owner_document_with_null :: "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "get_owner_document_with_null (Some ptr) = (do {
        document_ptr ← get_owner_document ptr;
```

```
        return (Some (cast_document_ptr2object_ptr document_ptr))})"
notation get_owner_document_with_null (<_ . ownerDocument>)

fun remove_with_null :: "((_::linorder) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "remove_with_null (Some child) = (case cast child of
      Some child_node ⇒ do {
        remove child_node;
        return (Some child)}
    | None ⇒ error NotFoundError)"
  | "remove_with_null None = error TypeError"
notation remove_with_null (<_ . remove'(')>)

fun remove_child_with_null ::
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "remove_child_with_null (Some ptr) (Some child) = (case cast child of
      Some child_node ⇒ do {
        remove_child ptr child_node;
        return (Some child)}
    | None ⇒ error NotFoundError)"
  | "remove_child_with_null None _ = error TypeError"
  | "remove_child_with_null _ None = error TypeError"
notation remove_child_with_null (<_ . removeChild>)

fun get_tag_name_with_null :: "((_) object_ptr option) ⇒ (_, attr_value) dom_prog"
  where
    "get_tag_name_with_null (Some ptr) = (case cast ptr of
      Some element_ptr ⇒ get_M element_ptr tag_name)"
notation get_tag_name_with_null (<_ . tagName>)

abbreviation "remove_attribute_with_null ptr k ≡ set_attribute_with_null2 ptr k None"
notation remove_attribute_with_null (<_ . removeAttribute'(_')>)

fun get_attribute_with_null :: "((_) object_ptr option) ⇒ attr_key ⇒ (_, attr_value option) dom_prog"
  where
    "get_attribute_with_null (Some ptr) k = (case cast ptr of
      Some element_ptr ⇒ get_attribute element_ptr k)"
fun get_attribute_with_null2 :: "((_) object_ptr option) ⇒ attr_key ⇒ (_, attr_value) dom_prog"
  where
    "get_attribute_with_null2 (Some ptr) k = (case cast ptr of
      Some element_ptr ⇒ do {
        a ← get_attribute element_ptr k;
        return (the a)})"
notation get_attribute_with_null (<_ . getAttribute'(_')>)
notation get_attribute_with_null2 (<_ . getAttribute'(_')>)

fun get_parent_with_null :: "((_::linorder) object_ptr option) ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "get_parent_with_null (Some ptr) = (case cast ptr of
      Some node_ptr ⇒ get_parent node_ptr)"
notation get_parent_with_null (<_ . parentNode>)

fun first_child_with_null :: "((_) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "first_child_with_null (Some ptr) = do {
      child_opt ← first_child ptr;
      return (case child_opt of
        Some child ⇒ Some (cast child)
      | None ⇒ None)}"
notation first_child_with_null (<_ . firstChild>)

fun adopt_node_with_null ::
```

```
  "((_::linorder) object_ptr option) ⇒ ((_) object_ptr option) ⇒ (_, ((_) object_ptr option)) dom_prog"
  where
    "adopt_node_with_null (Some ptr) (Some child) = (case cast ptr of
      Some document_ptr ⇒ (case cast child of
        Some child_node ⇒ do {
          adopt_node document_ptr child_node;
          return (Some child)}))"
notation adopt_node_with_null (<_ . adoptNode'(_')>)


fun get_shadow_root_with_null :: "((_) object_ptr option) ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "get_shadow_root_with_null (Some ptr) = (case cast ptr of
      Some element_ptr ⇒ do {
        shadow_root ← get_shadow_root element_ptr;
        (case shadow_root of Some sr ⇒ return (Some (cast sr))
                           | None ⇒ return None)})"
notation get_shadow_root_with_null (<_ . shadowRoot>)
```

3.1.2 Making the functions under test compatible with untyped languages such as JavaScript

```
fun get_element_by_id_si_with_null ::
  "(_::linorder) object_ptr option ⇒ string ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "get_element_by_id_si_with_null (Some ptr) id' = do {
        element_ptr_opt ← get_element_by_id_si ptr id';
        (case element_ptr_opt of
          Some element_ptr ⇒ return (Some (cast_element_ptr2object_ptr element_ptr))
        | None ⇒ return None)}"
  | "get_element_by_id_si_with_null _ _ = error SegmentationFault"

fun find_slot_closed_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option) dom_prog"
  where
    "find_slot_closed_with_null (Some ptr) = (case cast_object_ptr2node_ptr ptr of
      Some node_ptr ⇒ do {
        element_ptr_opt ← find_slot True node_ptr;
        (case element_ptr_opt of
          Some element_ptr ⇒ return (Some (cast_element_ptr2object_ptr element_ptr))
        | None ⇒ return None)}
    | None ⇒ error SegmentationFault)"
  | "find_slot_closed_with_null None = error SegmentationFault"
notation find_slot_closed_with_null (<_ . assignedSlot>)

fun assigned_nodes_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "assigned_nodes_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes element_ptr;
        return (map Some (map cast_node_ptr2object_ptr l))}
    | None ⇒ error SegmentationFault)"
  | "assigned_nodes_with_null None = error SegmentationFault"
notation assigned_nodes_with_null (<_ . assignedNodes'(')>)

fun assigned_nodes_flatten_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "assigned_nodes_flatten_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes_flatten element_ptr;
        return (map Some (map cast_node_ptr2object_ptr l))}
```

```
    | None ⇒ error SegmentationFault)"
  | "assigned_nodes_flatten_with_null None = error SegmentationFault"
notation assigned_nodes_flatten_with_null (<_ . assignedNodes'(True')>)

fun get_assigned_elements_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "get_assigned_elements_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes element_ptr;
        l ← map_filter_M (return ∘ cast_node_ptr2element_ptr) l;
        return (map Some (map cast l))}
    | None ⇒ error SegmentationFault)"
  | "get_assigned_elements_with_null None = error SegmentationFault"
notation get_assigned_elements_with_null (<_ . assignedElements'(')>)

fun get_assigned_elements_flatten_with_null ::
  "(_::linorder) object_ptr option ⇒ (_, (_) object_ptr option list) dom_prog"
  where
    "get_assigned_elements_flatten_with_null (Some ptr) = (case cast_object_ptr2element_ptr ptr of
      Some element_ptr ⇒ do {
        l ← assigned_nodes_flatten element_ptr;
        return (map Some (map cast_node_ptr2object_ptr l))}
    | None ⇒ error SegmentationFault)"
  | "get_assigned_elements_flatten_with_null None = error SegmentationFault"
notation get_assigned_elements_flatten_with_null (<_ . assignedElements'(True')>)

fun createTestTree ::
  "(_::linorder) object_ptr option ⇒ (_, string ⇒ (_, (_) object_ptr option) dom_prog) dom_prog"
  where
    "createTestTree (Some ref) = do {
      tups ← to_tree_order_si ref ⤜ map_filter_M (λptr. do {
        (case cast ptr of
          Some element_ptr ⇒ do {
            iden_opt ← get_attribute element_ptr ''id'';
            (case iden_opt of
              Some iden ⇒ return (Some (iden, ptr))
            | None ⇒ return None)
          }
        | None ⇒ return None)});
      return (return ∘ map_of tups)
    }"
  | "createTestTree None = error SegmentationFault"


end
```

3.2 Testing slots (slots)

This theory contains the test cases for slots.

```
theory slots
imports
  "Shadow_DOM_BaseTest"
begin

definition slots_heap :: "heap_final" where
  "slots_heap = create_heap [(cast (document_ptr.Ref 1), cast (create_document_obj html (Some (cast (element_ptr.Re
1))) [])),
    (cast (element_ptr.Ref 1), cast (create_element_obj ''html'' [cast (element_ptr.Ref 2), cast (element_ptr.Ref
8)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7)] 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 ''Shadow%20DOM%3A%20Slots%20and%20assignments
    (cast (element_ptr.Ref 4), cast (create_element_obj ''meta'' [] (fmap_of_list [(''name'', ''author''),
(''title'', ''Hayato Ito''), (''href'', ''mailto:hayato@google.com'')]) None)),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''resources/shadow-d
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''body'' [cast (element_ptr.Ref 9), cast (element_ptr.Ref
13), cast (element_ptr.Ref 14), cast (element_ptr.Ref 18), cast (element_ptr.Ref 19), cast (element_ptr.Ref
21), cast (element_ptr.Ref 22), cast (element_ptr.Ref 30), cast (element_ptr.Ref 31), cast (element_ptr.Ref
39), cast (element_ptr.Ref 40), cast (element_ptr.Ref 48), cast (element_ptr.Ref 49), cast (element_ptr.Ref
57), cast (element_ptr.Ref 58), cast (element_ptr.Ref 64), cast (element_ptr.Ref 65), cast (element_ptr.Ref
71), cast (element_ptr.Ref 72), cast (element_ptr.Ref 78), cast (element_ptr.Ref 79), cast (element_ptr.Ref
85), cast (element_ptr.Ref 86), cast (element_ptr.Ref 92), cast (element_ptr.Ref 93), cast (element_ptr.Ref
112)] fmempty None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''div'' [cast (element_ptr.Ref 10)] (fmap_of_list
[(''id'', ''test_basic'')]) None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''div'' [cast (element_ptr.Ref 11)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 1))))),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 1), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 12)])),
    (cast (element_ptr.Ref 12), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 13), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 14), cast (create_element_obj ''div'' [cast (element_ptr.Ref 15)] (fmap_of_list
[(''id'', ''test_basic_closed'')]) None)),
    (cast (element_ptr.Ref 15), cast (create_element_obj ''div'' [cast (element_ptr.Ref 16)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 2))))),
    (cast (element_ptr.Ref 16), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 2), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 17)])),
    (cast (element_ptr.Ref 17), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 18), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 19), cast (create_element_obj ''div'' [cast (element_ptr.Ref 20)] (fmap_of_list
[(''id'', ''test_slot_not_in_shadow'')]) None)),
    (cast (element_ptr.Ref 20), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1'')]) None)),
    (cast (element_ptr.Ref 21), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 4)] fmempty
None)),
    (cast (character_data_ptr.Ref 4), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 22), cast (create_element_obj ''div'' [cast (element_ptr.Ref 23), cast (element_ptr.Ref
25)] (fmap_of_list [(''id'', ''test_slot_not_in_shadow_2'')]) None)),
    (cast (element_ptr.Ref 23), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 24)] (fmap_of_list
[(''id'', ''s1'')]) None)),
    (cast (element_ptr.Ref 24), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1'')]) None)),
    (cast (element_ptr.Ref 25), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 26), cast (element_ptr.Ref
27)] (fmap_of_list [(''id'', ''s2'')]) None)),
    (cast (element_ptr.Ref 26), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2'')]) None)),
    (cast (element_ptr.Ref 27), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 28), cast (element_ptr.Ref
29)] (fmap_of_list [(''id'', ''s3'')]) None)),
    (cast (element_ptr.Ref 28), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3_1'')])
None)),
    (cast (element_ptr.Ref 29), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3_2'')])
None)),
```

```
    (cast (element_ptr.Ref 30), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 5)] fmempty
None)),
    (cast (character_data_ptr.Ref 5), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 31), cast (create_element_obj ''div'' [cast (element_ptr.Ref 32)] (fmap_of_list
[(''id'', ''test_slot_name_matching'')]) None)),
    (cast (element_ptr.Ref 32), cast (create_element_obj ''div'' [cast (element_ptr.Ref 33), cast (element_ptr.Ref
34), cast (element_ptr.Ref 35)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 3)))),
    (cast (element_ptr.Ref 33), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 34), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 35), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3''), (''slot'',
''yyy'')]) None)),
    (cast (shadow_root_ptr.Ref 3), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 36), cast (element_ptr.
37), cast (element_ptr.Ref 38)])),
    (cast (element_ptr.Ref 36), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 37), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 38), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s3''), (''name'',
''xxx'')]) None)),
    (cast (element_ptr.Ref 39), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 6)] fmempty
None)),
    (cast (character_data_ptr.Ref 6), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 40), cast (create_element_obj ''div'' [cast (element_ptr.Ref 41)] (fmap_of_list
[(''id'', ''test_no_direct_host_child'')]) None)),
    (cast (element_ptr.Ref 41), cast (create_element_obj ''div'' [cast (element_ptr.Ref 42), cast (element_ptr.Ref
43), cast (element_ptr.Ref 44)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 4)))),
    (cast (element_ptr.Ref 42), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 43), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 44), cast (create_element_obj ''div'' [cast (element_ptr.Ref 45)] fmempty None)),
    (cast (element_ptr.Ref 45), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 4), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 46), cast (element_ptr.
47)])),
    (cast (element_ptr.Ref 46), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 47), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 48), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 7)] fmempty
None)),
    (cast (character_data_ptr.Ref 7), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 49), cast (create_element_obj ''div'' [cast (element_ptr.Ref 50)] (fmap_of_list
[(''id'', ''test_default_slot'')]) None)),
    (cast (element_ptr.Ref 50), cast (create_element_obj ''div'' [cast (element_ptr.Ref 51), cast (element_ptr.Ref
52), cast (element_ptr.Ref 53)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 5)))),
    (cast (element_ptr.Ref 51), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1'')]) None)),
    (cast (element_ptr.Ref 52), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
'''')]) None)),
    (cast (element_ptr.Ref 53), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3''), (''slot'',
''foo'')]) None)),
    (cast (shadow_root_ptr.Ref 5), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 54), cast (element_ptr.
55), cast (element_ptr.Ref 56)])),
    (cast (element_ptr.Ref 54), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 55), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2'')]) None)),
    (cast (element_ptr.Ref 56), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s3'')]) None)),
    (cast (element_ptr.Ref 57), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 8)] fmempty
None)),
    (cast (character_data_ptr.Ref 8), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 58), cast (create_element_obj ''div'' [cast (element_ptr.Ref 59)] (fmap_of_list
```

```
[(''id'', ''test_slot_in_slot'')]) None)),
    (cast (element_ptr.Ref 59), cast (create_element_obj ''div'' [cast (element_ptr.Ref 60), cast (element_ptr.Ref
61)] (fmap_of_list [(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 6))))),
    (cast (element_ptr.Ref 60), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 61), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 6), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 62)])),
    (cast (element_ptr.Ref 62), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 63)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 63), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 64), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 9)] fmempty
None)),
    (cast (character_data_ptr.Ref 9), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 65), cast (create_element_obj ''div'' [cast (element_ptr.Ref 66)] (fmap_of_list
[(''id'', ''test_slot_is_assigned_to_slot'')]) None)),
    (cast (element_ptr.Ref 66), cast (create_element_obj ''div'' [cast (element_ptr.Ref 67)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 7))))),
    (cast (element_ptr.Ref 67), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 7), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 68)])),
    (cast (element_ptr.Ref 68), cast (create_element_obj ''div'' [cast (element_ptr.Ref 69)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 8))))),
    (cast (element_ptr.Ref 69), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 8), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 70)])),
    (cast (element_ptr.Ref 70), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 71), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 10)] fmempty
None)),
    (cast (character_data_ptr.Ref 10), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 72), cast (create_element_obj ''div'' [cast (element_ptr.Ref 73)] (fmap_of_list
[(''id'', ''test_open_closed'')]) None)),
    (cast (element_ptr.Ref 73), cast (create_element_obj ''div'' [cast (element_ptr.Ref 74)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 9))))),
    (cast (element_ptr.Ref 74), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 9), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 75)])),
    (cast (element_ptr.Ref 75), cast (create_element_obj ''div'' [cast (element_ptr.Ref 76)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 10))))),
    (cast (element_ptr.Ref 76), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 10), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 77)])),
    (cast (element_ptr.Ref 77), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 78), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 11)] fmempty
None)),
    (cast (character_data_ptr.Ref 11), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 79), cast (create_element_obj ''div'' [cast (element_ptr.Ref 80)] (fmap_of_list
[(''id'', ''test_closed_closed'')]) None)),
    (cast (element_ptr.Ref 80), cast (create_element_obj ''div'' [cast (element_ptr.Ref 81)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 11))))),
    (cast (element_ptr.Ref 81), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 11), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 82)])),
    (cast (element_ptr.Ref 82), cast (create_element_obj ''div'' [cast (element_ptr.Ref 83)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 12))))),
    (cast (element_ptr.Ref 83), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 12), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 84)])),
    (cast (element_ptr.Ref 84), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
```

```
    (cast (element_ptr.Ref 85), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 12)] fmempty
None)),
    (cast (character_data_ptr.Ref 12), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 86), cast (create_element_obj ''div'' [cast (element_ptr.Ref 87)] (fmap_of_list
[(''id'', ''test_closed_open'')]) None)),
    (cast (element_ptr.Ref 87), cast (create_element_obj ''div'' [cast (element_ptr.Ref 88)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 13))))),
    (cast (element_ptr.Ref 88), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 13), cast (create_shadow_root_obj Closed [cast (element_ptr.Ref 89)])),
    (cast (element_ptr.Ref 89), cast (create_element_obj ''div'' [cast (element_ptr.Ref 90)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 14))))),
    (cast (element_ptr.Ref 90), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 14), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 91)])),
    (cast (element_ptr.Ref 91), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 92), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 13)] fmempty
None)),
    (cast (character_data_ptr.Ref 13), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 93), cast (create_element_obj ''div'' [cast (element_ptr.Ref 94)] (fmap_of_list
[(''id'', ''test_complex'')]) None)),
    (cast (element_ptr.Ref 94), cast (create_element_obj ''div'' [cast (element_ptr.Ref 95), cast (element_ptr.Ref
96), cast (element_ptr.Ref 97), cast (element_ptr.Ref 98)] (fmap_of_list [(''id'', ''host1'')]) (Some (cast
(shadow_root_ptr.Ref 15))))),
    (cast (element_ptr.Ref 95), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (element_ptr.Ref 96), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c2''), (''slot'',
''slot2'')]) None)),
    (cast (element_ptr.Ref 97), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c3'')]) None)),
    (cast (element_ptr.Ref 98), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c4''), (''slot'',
''slot-none'')]) None)),
    (cast (shadow_root_ptr.Ref 15), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 99)])),
    (cast (element_ptr.Ref 99), cast (create_element_obj ''div'' [cast (element_ptr.Ref 100), cast (element_ptr.Ref
101), cast (element_ptr.Ref 102), cast (element_ptr.Ref 103), cast (element_ptr.Ref 104), cast (element_ptr.Ref
105), cast (element_ptr.Ref 106), cast (element_ptr.Ref 107)] (fmap_of_list [(''id'', ''host2'')]) (Some
(cast (shadow_root_ptr.Ref 16))))),
    (cast (element_ptr.Ref 100), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s1''), (''name'',
''slot1''), (''slot'', ''slot5'')]) None)),
    (cast (element_ptr.Ref 101), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s2''), (''name'',
''slot2''), (''slot'', ''slot6'')]) None)),
    (cast (element_ptr.Ref 102), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s3'')])
None)),
    (cast (element_ptr.Ref 103), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s4''), (''name'',
''slot4''), (''slot'', ''slot-none'')]) None)),
    (cast (element_ptr.Ref 104), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c5''), (''slot'',
''slot5'')]) None)),
    (cast (element_ptr.Ref 105), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c6''), (''slot'',
''slot6'')]) None)),
    (cast (element_ptr.Ref 106), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c7'')]) None)),
    (cast (element_ptr.Ref 107), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c8''), (''slot'',
''slot-none'')]) None)),
    (cast (shadow_root_ptr.Ref 16), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 108), cast
(element_ptr.Ref 109), cast (element_ptr.Ref 110), cast (element_ptr.Ref 111)])),
    (cast (element_ptr.Ref 108), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s5''), (''name'',
''slot5'')]) None)),
    (cast (element_ptr.Ref 109), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s6''), (''name'',
''slot6'')]) None)),
    (cast (element_ptr.Ref 110), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s7'')])
None)),
    (cast (element_ptr.Ref 111), cast (create_element_obj ''slot'' [] (fmap_of_list [(''id'', ''s8''), (''name'',
''slot8'')]) None)),
    (cast (element_ptr.Ref 112), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 14)]
```

```
fmempty None)),
    (cast (character_data_ptr.Ref 14), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

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

'Slots: Basic.'

lemma *"test (do {*
```
  tmp0 ← slots_document . getElementById(''test_basic'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_basic'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedNodes();
  tmp7 ← n . ''c1'';
  assert_array_equals(tmp6, [tmp7])
}) slots_heap"
```
⟨*proof*⟩

'Slots: Basic, elements only.'

lemma *"test (do {*
```
  tmp0 ← slots_document . getElementById(''test_basic'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  tmp3 ← n . ''c1'';
  assert_array_equals(tmp2, [tmp3])
}) slots_heap"
```
⟨*proof*⟩

'Slots: Slots in closed.'

lemma *"test (do {*
```
  tmp0 ← slots_document . getElementById(''test_basic_closed'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_basic_closed'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedNodes();
  tmp6 ← n . ''c1'';
  assert_array_equals(tmp5, [tmp6])
}) slots_heap"
```
⟨*proof*⟩

'Slots: Slots in closed, elements only.'

lemma *"test (do {*
```
  tmp0 ← slots_document . getElementById(''test_basic_closed'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  tmp3 ← n . ''c1'';
  assert_array_equals(tmp2, [tmp3])
}) slots_heap"
```
⟨*proof*⟩

'Slots: Slots not in a shadow tree.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_slot_not_in_shadow'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_not_in_shadow'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp3 ← tmp2 . assignedNodes();
  assert_array_equals(tmp3, [])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Slots not in a shadow tree, elements only.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_slot_not_in_shadow'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  assert_array_equals(tmp2, [])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Distributed nodes for Slots not in a shadow tree.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_slot_not_in_shadow_2'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_not_in_shadow_2'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''c2'';
  tmp5 ← tmp4 . assignedSlot;
  assert_equals(tmp5, None);
  tmp6 ← n . ''c3_1'';
  tmp7 ← tmp6 . assignedSlot;
  assert_equals(tmp7, None);
  tmp8 ← n . ''c3_2'';
  tmp9 ← tmp8 . assignedSlot;
  assert_equals(tmp9, None);
  tmp10 ← n . ''s1'';
  tmp11 ← tmp10 . assignedNodes();
  assert_array_equals(tmp11, []);
  tmp12 ← n . ''s2'';
  tmp13 ← tmp12 . assignedNodes();
  assert_array_equals(tmp13, []);
  tmp14 ← n . ''s3'';
  tmp15 ← tmp14 . assignedNodes();
  assert_array_equals(tmp15, []);
  tmp16 ← n . ''s1'';
  tmp17 ← tmp16 . assignedNodes(True);
  assert_array_equals(tmp17, []);
  tmp18 ← n . ''s2'';
  tmp19 ← tmp18 . assignedNodes(True);
  assert_array_equals(tmp19, []);
  tmp20 ← n . ''s3'';
  tmp21 ← tmp20 . assignedNodes(True);
  assert_array_equals(tmp21, [])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Name matching'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_slot_name_matching'');
  n ← createTestTree(tmp0);
```

```
  tmp1 ← n . ''test_slot_name_matching'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  assert_equals(tmp9, None)
}) slots_heap"
```

⟨*proof*⟩

'Slots: No direct host child.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_no_direct_host_child'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_no_direct_host_child'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s1'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  assert_equals(tmp9, None);
  tmp10 ← n . ''s1'';
  tmp11 ← tmp10 . assignedNodes();
  tmp12 ← n . ''c1'';
  tmp13 ← n . ''c2'';
  assert_array_equals(tmp11, [tmp12, tmp13])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Default Slot.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_default_slot'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_default_slot'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s2'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  assert_equals(tmp9, None)
}) slots_heap"
```

⟨*proof*⟩

'Slots: Slot in Slot does not matter in assignment.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_slot_in_slot'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_in_slot'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s2'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s1'';
  assert_equals(tmp6, tmp7)
}) slots_heap"
```

⟨*proof*⟩

'Slots: Slot is assigned to another slot'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_slot_is_assigned_to_slot'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_slot_is_assigned_to_slot'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''s1'';
  tmp9 ← tmp8 . assignedNodes();
  tmp10 ← n . ''c1'';
  assert_array_equals(tmp9, [tmp10]);
  tmp11 ← n . ''s2'';
  tmp12 ← tmp11 . assignedNodes();
  tmp13 ← n . ''s1'';
  assert_array_equals(tmp12, [tmp13]);
  tmp14 ← n . ''s1'';
  tmp15 ← tmp14 . assignedNodes(True);
  tmp16 ← n . ''c1'';
  assert_array_equals(tmp15, [tmp16]);
  tmp17 ← n . ''s2'';
  tmp18 ← tmp17 . assignedNodes(True);
  tmp19 ← n . ''c1'';
  assert_array_equals(tmp18, [tmp19])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Open > Closed.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_open_closed'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_open_closed'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
```

```
  tmp9 ← n . ''c1'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s2'';
  tmp11 ← tmp10 . assignedNodes();
  tmp12 ← n . ''s1'';
  assert_array_equals(tmp11, [tmp12]);
  tmp13 ← n . ''s1'';
  tmp14 ← tmp13 . assignedNodes(True);
  tmp15 ← n . ''c1'';
  assert_array_equals(tmp14, [tmp15]);
  tmp16 ← n . ''s2'';
  tmp17 ← tmp16 . assignedNodes(True);
  tmp18 ← n . ''c1'';
  assert_array_equals(tmp17, [tmp18])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Closed > Closed.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_closed_closed'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_closed_closed'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedSlot;
  assert_equals(tmp5, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp6 ← n . ''s1'';
  tmp7 ← tmp6 . assignedNodes();
  tmp8 ← n . ''c1'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s2'';
  tmp10 ← tmp9 . assignedNodes();
  tmp11 ← n . ''s1'';
  assert_array_equals(tmp10, [tmp11]);
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  assert_array_equals(tmp16, [tmp17])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Closed > Open.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_closed_open'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_closed_open'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None, ''A slot in a closed shadow tree should not be accessed via assignedSlot'');
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedSlot;
  tmp6 ← n . ''s2'';
  assert_equals(tmp5, tmp6);
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
```

```
  tmp9 ← n . ''c1'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s2'';
  tmp11 ← tmp10 . assignedNodes();
  tmp12 ← n . ''s1'';
  assert_array_equals(tmp11, [tmp12]);
  tmp13 ← n . ''s1'';
  tmp14 ← tmp13 . assignedNodes(True);
  tmp15 ← n . ''c1'';
  assert_array_equals(tmp14, [tmp15]);
  tmp16 ← n . ''s2'';
  tmp17 ← tmp16 . assignedNodes(True);
  tmp18 ← n . ''c1'';
  assert_array_equals(tmp17, [tmp18])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Complex case: Basi line.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''c2'';
  tmp6 ← tmp5 . assignedSlot;
  tmp7 ← n . ''s2'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''c3'';
  tmp9 ← tmp8 . assignedSlot;
  tmp10 ← n . ''s3'';
  assert_equals(tmp9, tmp10);
  tmp11 ← n . ''c4'';
  tmp12 ← tmp11 . assignedSlot;
  assert_equals(tmp12, None);
  tmp13 ← n . ''s1'';
  tmp14 ← tmp13 . assignedSlot;
  tmp15 ← n . ''s5'';
  assert_equals(tmp14, tmp15);
  tmp16 ← n . ''s2'';
  tmp17 ← tmp16 . assignedSlot;
  tmp18 ← n . ''s6'';
  assert_equals(tmp17, tmp18);
  tmp19 ← n . ''s3'';
  tmp20 ← tmp19 . assignedSlot;
  tmp21 ← n . ''s7'';
  assert_equals(tmp20, tmp21);
  tmp22 ← n . ''s4'';
  tmp23 ← tmp22 . assignedSlot;
  assert_equals(tmp23, None);
  tmp24 ← n . ''c5'';
  tmp25 ← tmp24 . assignedSlot;
  tmp26 ← n . ''s5'';
  assert_equals(tmp25, tmp26);
  tmp27 ← n . ''c6'';
  tmp28 ← tmp27 . assignedSlot;
  tmp29 ← n . ''s6'';
  assert_equals(tmp28, tmp29);
  tmp30 ← n . ''c7'';
  tmp31 ← tmp30 . assignedSlot;
```

```
tmp32 ← n . ''s7'';
assert_equals(tmp31, tmp32);
tmp33 ← n . ''c8'';
tmp34 ← tmp33 . assignedSlot;
assert_equals(tmp34, None);
tmp35 ← n . ''s1'';
tmp36 ← tmp35 . assignedNodes();
tmp37 ← n . ''c1'';
assert_array_equals(tmp36, [tmp37]);
tmp38 ← n . ''s2'';
tmp39 ← tmp38 . assignedNodes();
tmp40 ← n . ''c2'';
assert_array_equals(tmp39, [tmp40]);
tmp41 ← n . ''s3'';
tmp42 ← tmp41 . assignedNodes();
tmp43 ← n . ''c3'';
assert_array_equals(tmp42, [tmp43]);
tmp44 ← n . ''s4'';
tmp45 ← tmp44 . assignedNodes();
assert_array_equals(tmp45, []);
tmp46 ← n . ''s5'';
tmp47 ← tmp46 . assignedNodes();
tmp48 ← n . ''s1'';
tmp49 ← n . ''c5'';
assert_array_equals(tmp47, [tmp48, tmp49]);
tmp50 ← n . ''s6'';
tmp51 ← tmp50 . assignedNodes();
tmp52 ← n . ''s2'';
tmp53 ← n . ''c6'';
assert_array_equals(tmp51, [tmp52, tmp53]);
tmp54 ← n . ''s7'';
tmp55 ← tmp54 . assignedNodes();
tmp56 ← n . ''s3'';
tmp57 ← n . ''c7'';
assert_array_equals(tmp55, [tmp56, tmp57]);
tmp58 ← n . ''s8'';
tmp59 ← tmp58 . assignedNodes();
assert_array_equals(tmp59, []);
tmp60 ← n . ''s1'';
tmp61 ← tmp60 . assignedNodes(True);
tmp62 ← n . ''c1'';
assert_array_equals(tmp61, [tmp62]);
tmp63 ← n . ''s2'';
tmp64 ← tmp63 . assignedNodes(True);
tmp65 ← n . ''c2'';
assert_array_equals(tmp64, [tmp65]);
tmp66 ← n . ''s3'';
tmp67 ← tmp66 . assignedNodes(True);
tmp68 ← n . ''c3'';
assert_array_equals(tmp67, [tmp68]);
tmp69 ← n . ''s4'';
tmp70 ← tmp69 . assignedNodes(True);
assert_array_equals(tmp70, []);
tmp71 ← n . ''s5'';
tmp72 ← tmp71 . assignedNodes(True);
tmp73 ← n . ''c1'';
tmp74 ← n . ''c5'';
assert_array_equals(tmp72, [tmp73, tmp74]);
tmp75 ← n . ''s6'';
tmp76 ← tmp75 . assignedNodes(True);
tmp77 ← n . ''c2'';
tmp78 ← n . ''c6'';
assert_array_equals(tmp76, [tmp77, tmp78]);
```

```
  tmp79 ← n . ''s7'';
  tmp80 ← tmp79 . assignedNodes(True);
  tmp81 ← n . ''c3'';
  tmp82 ← n . ''c7'';
  assert_array_equals(tmp80, [tmp81, tmp82]);
  tmp83 ← n . ''s8'';
  tmp84 ← tmp83 . assignedNodes(True);
  assert_array_equals(tmp84, [])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: appendChild.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  d1 ← slots_document . createElement(''div'');
  d1 . setAttribute(''slot'', ''slot1'');
  tmp2 ← n . ''host1'';
  tmp2 . appendChild(d1);
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5, d1]);
  tmp6 ← d1 . assignedSlot;
  tmp7 ← n . ''s1'';
  assert_equals(tmp6, tmp7);
  tmp8 ← n . ''s5'';
  tmp9 ← tmp8 . assignedNodes(True);
  tmp10 ← n . ''c1'';
  tmp11 ← n . ''c5'';
  assert_array_equals(tmp9, [tmp10, d1, tmp11])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Change slot= attribute 1.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp2 . setAttribute(''slot'', ''slot-none'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''c1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s5'';
  tmp8 ← tmp7 . assignedNodes(True);
  tmp9 ← n . ''c5'';
  assert_array_equals(tmp8, [tmp9])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Change slot= attribute 2.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
```

```
  tmp2 ← n . ''c1'';
  tmp2 . setAttribute(''slot'', ''slot2'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s2'';
  tmp6 ← tmp5 . assignedNodes();
  tmp7 ← n . ''c1'';
  tmp8 ← n . ''c2'';
  assert_array_equals(tmp6, [tmp7, tmp8]);
  tmp9 ← n . ''c1'';
  tmp10 ← tmp9 . assignedSlot;
  tmp11 ← n . ''s2'';
  assert_equals(tmp10, tmp11);
  tmp12 ← n . ''s5'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c5'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s6'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  tmp18 ← n . ''c2'';
  tmp19 ← n . ''c6'';
  assert_array_equals(tmp16, [tmp17, tmp18, tmp19])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Change slot= attribute 3.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c4'';
  tmp2 . setAttribute(''slot'', ''slot1'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  tmp5 ← n . ''c1'';
  tmp6 ← n . ''c4'';
  assert_array_equals(tmp4, [tmp5, tmp6]);
  tmp7 ← n . ''c4'';
  tmp8 ← tmp7 . assignedSlot;
  tmp9 ← n . ''s1'';
  assert_equals(tmp8, tmp9);
  tmp10 ← n . ''s5'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''c1'';
  tmp13 ← n . ''c4'';
  tmp14 ← n . ''c5'';
  assert_array_equals(tmp11, [tmp12, tmp13, tmp14])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Remove a child.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
```

```
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''c1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s5'';
  tmp8 ← tmp7 . assignedNodes(True);
  tmp9 ← n . ''c5'';
  assert_array_equals(tmp8, [tmp9])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Add a slot: after.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  slot ← slots_document . createElement(''slot'');
  slot . setAttribute(''name'', ''slot1'');
  tmp2 ← n . ''host2'';
  tmp2 . appendChild(slot);
  tmp3 ← slot . assignedNodes();
  assert_array_equals(tmp3, [])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Add a slot: before.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  slot ← slots_document . createElement(''slot'');
  slot . setAttribute(''name'', ''slot1'');
  tmp3 ← n . ''s1'';
  tmp2 ← n . ''host2'';
  tmp2 . insertBefore(slot, tmp3);
  tmp4 ← slot . assignedNodes();
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''c1'';
  tmp7 ← tmp6 . assignedSlot;
  assert_equals(tmp7, slot);
  tmp8 ← n . ''s7'';
  tmp9 ← tmp8 . assignedNodes();
  tmp10 ← n . ''s3'';
  tmp11 ← n . ''c7'';
  assert_array_equals(tmp9, [slot, tmp10, tmp11]);
  tmp12 ← n . ''s7'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  tmp15 ← n . ''c3'';
  tmp16 ← n . ''c7'';
  assert_array_equals(tmp13, [tmp14, tmp15, tmp16])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Remove a slot.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
```

```
  tmp2 ← n . ''s1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''c1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s5'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''c5'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s5'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''c5'';
  assert_array_equals(tmp11, [tmp12])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Change slot name= attribute.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp2 . setAttribute(''name'', ''slot2'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  tmp5 ← n . ''c2'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''c1'';
  tmp7 ← tmp6 . assignedSlot;
  assert_equals(tmp7, None);
  tmp8 ← n . ''c2'';
  tmp9 ← tmp8 . assignedSlot;
  tmp10 ← n . ''s1'';
  assert_equals(tmp9, tmp10);
  tmp11 ← n . ''s5'';
  tmp12 ← tmp11 . assignedNodes();
  tmp13 ← n . ''s1'';
  tmp14 ← n . ''c5'';
  assert_array_equals(tmp12, [tmp13, tmp14]);
  tmp15 ← n . ''s5'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c2'';
  tmp18 ← n . ''c5'';
  assert_array_equals(tmp16, [tmp17, tmp18])
}) slots_heap"
```

⟨*proof*⟩

'Slots: Mutation: Change slot slot= attribute.'

lemma *"test (do {*

```
  tmp0 ← slots_document . getElementById(''test_complex'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test_complex'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp2 . setAttribute(''slot'', ''slot6'');
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
```

```
  tmp6 ← n . ''s5'';
  tmp7 ← tmp6 . assignedNodes();
  tmp8 ← n . ''c5'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s6'';
  tmp10 ← tmp9 . assignedNodes();
  tmp11 ← n . ''s1'';
  tmp12 ← n . ''s2'';
  tmp13 ← n . ''c6'';
  assert_array_equals(tmp10, [tmp11, tmp12, tmp13]);
  tmp14 ← n . ''s6'';
  tmp15 ← tmp14 . assignedNodes(True);
  tmp16 ← n . ''c1'';
  tmp17 ← n . ''c2'';
  tmp18 ← n . ''c6'';
  assert_array_equals(tmp15, [tmp16, tmp17, tmp18])
}) slots_heap"
```

⟨*proof*⟩

end

3.3 Testing slots_fallback (slots_fallback)

This theory contains the test cases for slots_fallback.

theory *slots_fallback*
imports
"Shadow_DOM_BaseTest"
begin

definition *slots_fallback_heap :: "heap$_{final}$"* **where**

```
  "slots_fallback_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
8)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7)] 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 ''Shadow%20DOM%3A%20Slots%20and%20fallback%20
    (cast (element_ptr.Ref 4), cast (create_element_obj ''meta'' [] (fmap_of_list [(''name'', ''author''),
(''title'', ''Hayato Ito''), (''href'', ''mailto:hayato@google.com'')]) None)),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''resources/shadow-d
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''body'' [cast (element_ptr.Ref 9), cast (element_ptr.Ref
13), cast (element_ptr.Ref 14), cast (element_ptr.Ref 19), cast (element_ptr.Ref 20), cast (element_ptr.Ref
26), cast (element_ptr.Ref 27), cast (element_ptr.Ref 33), cast (element_ptr.Ref 34), cast (element_ptr.Ref
46)] fmempty None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''div'' [cast (element_ptr.Ref 10)] (fmap_of_list
[(''id'', ''test1'')]) None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''host'')])
(Some (cast (shadow_root_ptr.Ref 1))))),
    (cast (shadow_root_ptr.Ref 1), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 11)])),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 12)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 12), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 13), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
```

```
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 14), cast (create_element_obj ''div'' [cast (element_ptr.Ref 15)] (fmap_of_list
[(''id'', ''test2'')]) None)),
    (cast (element_ptr.Ref 15), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''host'')])
(Some (cast (shadow_root_ptr.Ref 2)))),
    (cast (shadow_root_ptr.Ref 2), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 16)])),
    (cast (element_ptr.Ref 16), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 17)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 17), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 18)] (fmap_of_list
[(''id'', ''s2''), (''name'', ''slot2'')]) None)),
    (cast (element_ptr.Ref 18), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 19), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 20), cast (create_element_obj ''div'' [cast (element_ptr.Ref 21)] (fmap_of_list
[(''id'', ''test3'')]) None)),
    (cast (element_ptr.Ref 21), cast (create_element_obj ''div'' [cast (element_ptr.Ref 22)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 3)))),
    (cast (element_ptr.Ref 22), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 3), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 23)])),
    (cast (element_ptr.Ref 23), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 24)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 24), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 25)] (fmap_of_list
[(''id'', ''s2''), (''name'', ''slot2'')]) None)),
    (cast (element_ptr.Ref 25), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 26), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 4)] fmempty
None)),
    (cast (character_data_ptr.Ref 4), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 27), cast (create_element_obj ''div'' [cast (element_ptr.Ref 28)] (fmap_of_list
[(''id'', ''test4'')]) None)),
    (cast (element_ptr.Ref 28), cast (create_element_obj ''div'' [cast (element_ptr.Ref 29)] (fmap_of_list
[(''id'', ''host'')]) (Some (cast (shadow_root_ptr.Ref 4)))),
    (cast (element_ptr.Ref 29), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot2'')]) None)),
    (cast (shadow_root_ptr.Ref 4), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 30)])),
    (cast (element_ptr.Ref 30), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 31)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 31), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 32)] (fmap_of_list
[(''id'', ''s2''), (''name'', ''slot2'')]) None)),
    (cast (element_ptr.Ref 32), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 33), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 5)] fmempty
None)),
    (cast (character_data_ptr.Ref 5), cast (create_character_data_obj ''%3C%3Cscript%3E%3E'')),
    (cast (element_ptr.Ref 34), cast (create_element_obj ''div'' [cast (element_ptr.Ref 35)] (fmap_of_list
[(''id'', ''test5'')]) None)),
    (cast (element_ptr.Ref 35), cast (create_element_obj ''div'' [cast (element_ptr.Ref 36)] (fmap_of_list
[(''id'', ''host1'')]) (Some (cast (shadow_root_ptr.Ref 5)))),
    (cast (element_ptr.Ref 36), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''c1''), (''slot'',
''slot1'')]) None)),
    (cast (shadow_root_ptr.Ref 5), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 37)])),
    (cast (element_ptr.Ref 37), cast (create_element_obj ''div'' [cast (element_ptr.Ref 38)] (fmap_of_list
[(''id'', ''host2'')]) (Some (cast (shadow_root_ptr.Ref 6)))),
    (cast (element_ptr.Ref 38), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 39), cast (element_ptr.Ref
41)] (fmap_of_list [(''id'', ''s2''), (''name'', ''slot2''), (''slot'', ''slot3'')]) None)),
    (cast (element_ptr.Ref 39), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 40)] (fmap_of_list
[(''id'', ''s1''), (''name'', ''slot1'')]) None)),
    (cast (element_ptr.Ref 40), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f1'')]) None)),
    (cast (element_ptr.Ref 41), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f2'')]) None)),
    (cast (shadow_root_ptr.Ref 6), cast (create_shadow_root_obj Open [cast (element_ptr.Ref 42)])),
    (cast (element_ptr.Ref 42), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 43), cast (element_ptr.Ref
45)] (fmap_of_list [(''id'', ''s4''), (''name'', ''slot4'')]) None)),
    (cast (element_ptr.Ref 43), cast (create_element_obj ''slot'' [cast (element_ptr.Ref 44)] (fmap_of_list
```

```
[(''id'', ''s3''), (''name'', ''slot3'')]) None)),
    (cast (element_ptr.Ref 44), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f3'')]) None)),
    (cast (element_ptr.Ref 45), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''f4'')]) None)),
    (cast (element_ptr.Ref 46), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 6)] fmempty
None)),
    (cast (character_data_ptr.Ref 6), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"

definition slots_fallback_document :: "(unit, unit, unit, unit, unit, unit) object_ptr option" where "slots_fallba
= Some (cast (document_ptr.Ref 1))"
```

'Slots fallback: Basic.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test1'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test1'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''f1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedNodes();
  assert_array_equals(tmp5, []);
  tmp6 ← n . ''s1'';
  tmp7 ← tmp6 . assignedNodes(True);
  tmp8 ← n . ''f1'';
  assert_array_equals(tmp7, [tmp8])
}) slots_fallback_heap"
```

⟨*proof*⟩

'Slots fallback: Basic, elements only.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test1'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  assert_array_equals(tmp2, []);
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedElements(True);
  tmp5 ← n . ''f1'';
  assert_array_equals(tmp4, [tmp5])
}) slots_fallback_heap"
```

⟨*proof*⟩

'Slots fallback: Slots in Slots.'

```
lemma "test (do {
  tmp0 ← slots_fallback_document . getElementById(''test2'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test2'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''f1'';
  tmp3 ← tmp2 . assignedSlot;
  assert_equals(tmp3, None);
  tmp4 ← n . ''s1'';
  tmp5 ← tmp4 . assignedNodes();
  assert_array_equals(tmp5, []);
  tmp6 ← n . ''s2'';
  tmp7 ← tmp6 . assignedNodes();
  assert_array_equals(tmp7, []);
  tmp8 ← n . ''s1'';
  tmp9 ← tmp8 . assignedNodes(True);
  tmp10 ← n . ''f1'';
  assert_array_equals(tmp9, [tmp10]);
  tmp11 ← n . ''s2'';
```

```
  tmp12 ← tmp11 . assignedNodes(True);
  tmp13 ← n . ''f1'';
  assert_array_equals(tmp12, [tmp13])
}) slots_fallback_heap"
```
⟨*proof*⟩

'Slots fallback: Slots in Slots, elements only.'

lemma *"test (do {*
```
  tmp0 ← slots_fallback_document . getElementById(''test2'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  assert_array_equals(tmp2, []);
  tmp3 ← n . ''s2'';
  tmp4 ← tmp3 . assignedElements();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedElements(True);
  tmp7 ← n . ''f1'';
  assert_array_equals(tmp6, [tmp7]);
  tmp8 ← n . ''s2'';
  tmp9 ← tmp8 . assignedElements(True);
  tmp10 ← n . ''f1'';
  assert_array_equals(tmp9, [tmp10])
}) slots_fallback_heap"
```
⟨*proof*⟩

'Slots fallback: Fallback contents should not be used if a node is assigned.'

lemma *"test (do {*
```
  tmp0 ← slots_fallback_document . getElementById(''test3'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test3'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s1'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''f1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''c1'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s2'';
  tmp11 ← tmp10 . assignedNodes();
  assert_array_equals(tmp11, []);
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''f1'';
  assert_array_equals(tmp16, [tmp17])
}) slots_fallback_heap"
```
⟨*proof*⟩

'Slots fallback: Slots in Slots: Assigned nodes should be used as fallback contents of another slot'

lemma *"test (do {*
```
  tmp0 ← slots_fallback_document . getElementById(''test4'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test4'';
```

```
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp3 ← tmp2 . assignedSlot;
  tmp4 ← n . ''s2'';
  assert_equals(tmp3, tmp4);
  tmp5 ← n . ''f1'';
  tmp6 ← tmp5 . assignedSlot;
  assert_equals(tmp6, None);
  tmp7 ← n . ''s1'';
  tmp8 ← tmp7 . assignedNodes();
  assert_array_equals(tmp8, []);
  tmp9 ← n . ''s2'';
  tmp10 ← tmp9 . assignedNodes();
  tmp11 ← n . ''c1'';
  assert_array_equals(tmp10, [tmp11]);
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  assert_array_equals(tmp16, [tmp17])
}) slots_fallback_heap"
```

⟨*proof*⟩

'Slots fallback: Complex case.'

lemma *"test (do {*

```
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp3 ← tmp2 . assignedNodes();
  tmp4 ← n . ''c1'';
  assert_array_equals(tmp3, [tmp4]);
  tmp5 ← n . ''s2'';
  tmp6 ← tmp5 . assignedNodes();
  assert_array_equals(tmp6, []);
  tmp7 ← n . ''s3'';
  tmp8 ← tmp7 . assignedNodes();
  tmp9 ← n . ''s2'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s4'';
  tmp11 ← tmp10 . assignedNodes();
  assert_array_equals(tmp11, []);
  tmp12 ← n . ''s1'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  assert_array_equals(tmp13, [tmp14]);
  tmp15 ← n . ''s2'';
  tmp16 ← tmp15 . assignedNodes(True);
  tmp17 ← n . ''c1'';
  tmp18 ← n . ''f2'';
  assert_array_equals(tmp16, [tmp17, tmp18]);
  tmp19 ← n . ''s3'';
  tmp20 ← tmp19 . assignedNodes(True);
  tmp21 ← n . ''c1'';
  tmp22 ← n . ''f2'';
  assert_array_equals(tmp20, [tmp21, tmp22]);
  tmp23 ← n . ''s4'';
  tmp24 ← tmp23 . assignedNodes(True);
  tmp25 ← n . ''c1'';
```

```
  tmp26 ← n . ''f2'';
  tmp27 ← n . ''f4'';
  assert_array_equals(tmp24, [tmp25, tmp26, tmp27])
}) slots_fallback_heap"
```
⟨*proof*⟩

'Slots fallback: Complex case, elements only.'

lemma *"test (do {*
```
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''s1'';
  tmp2 ← tmp1 . assignedElements();
  tmp3 ← n . ''c1'';
  assert_array_equals(tmp2, [tmp3]);
  tmp4 ← n . ''s2'';
  tmp5 ← tmp4 . assignedElements();
  assert_array_equals(tmp5, []);
  tmp6 ← n . ''s3'';
  tmp7 ← tmp6 . assignedElements();
  tmp8 ← n . ''s2'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s4'';
  tmp10 ← tmp9 . assignedElements();
  assert_array_equals(tmp10, []);
  tmp11 ← n . ''s1'';
  tmp12 ← tmp11 . assignedElements(True);
  tmp13 ← n . ''c1'';
  assert_array_equals(tmp12, [tmp13]);
  tmp14 ← n . ''s2'';
  tmp15 ← tmp14 . assignedElements(True);
  tmp16 ← n . ''c1'';
  tmp17 ← n . ''f2'';
  assert_array_equals(tmp15, [tmp16, tmp17]);
  tmp18 ← n . ''s3'';
  tmp19 ← tmp18 . assignedElements(True);
  tmp20 ← n . ''c1'';
  tmp21 ← n . ''f2'';
  assert_array_equals(tmp19, [tmp20, tmp21]);
  tmp22 ← n . ''s4'';
  tmp23 ← tmp22 . assignedElements(True);
  tmp24 ← n . ''c1'';
  tmp25 ← n . ''f2'';
  tmp26 ← n . ''f4'';
  assert_array_equals(tmp23, [tmp24, tmp25, tmp26])
}) slots_fallback_heap"
```
⟨*proof*⟩

'Slots fallback: Mutation. Append fallback contents.'

lemma *"test (do {*
```
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  d1 ← slots_fallback_document . createElement(''div'');
  tmp2 ← n . ''s2'';
  tmp2 . appendChild(d1);
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes(True);
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''s2'';
  tmp7 ← tmp6 . assignedNodes(True);
  tmp8 ← n . ''c1'';
```

```
  tmp9 ← n . ''f2'';
  assert_array_equals(tmp7, [tmp8, tmp9, d1]);
  tmp10 ← n . ''s3'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''c1'';
  tmp13 ← n . ''f2'';
  assert_array_equals(tmp11, [tmp12, tmp13, d1]);
  tmp14 ← n . ''s4'';
  tmp15 ← tmp14 . assignedNodes(True);
  tmp16 ← n . ''c1'';
  tmp17 ← n . ''f2'';
  tmp18 ← n . ''f4'';
  assert_array_equals(tmp15, [tmp16, tmp17, d1, tmp18])
}) slots_fallback_heap"
```

⟨*proof*⟩

'Slots fallback: Mutation. Remove fallback contents.'

lemma *"test (do {*

```
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''f2'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes(True);
  tmp5 ← n . ''c1'';
  assert_array_equals(tmp4, [tmp5]);
  tmp6 ← n . ''s2'';
  tmp7 ← tmp6 . assignedNodes(True);
  tmp8 ← n . ''c1'';
  assert_array_equals(tmp7, [tmp8]);
  tmp9 ← n . ''s3'';
  tmp10 ← tmp9 . assignedNodes(True);
  tmp11 ← n . ''c1'';
  assert_array_equals(tmp10, [tmp11]);
  tmp12 ← n . ''s4'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''c1'';
  tmp15 ← n . ''f4'';
  assert_array_equals(tmp13, [tmp14, tmp15])
}) slots_fallback_heap"
```

⟨*proof*⟩

'Slots fallback: Mutation. Assign a node to a slot so that fallback contens are no longer used.'

lemma *"test (do {*

```
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  d2 ← slots_fallback_document . createElement(''div'');
  d2 . setAttribute(''slot'', ''slot2'');
  tmp2 ← n . ''host1'';
  tmp2 . appendChild(d2);
  tmp3 ← n . ''s2'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, [d2]);
  tmp5 ← n . ''s2'';
  tmp6 ← tmp5 . assignedNodes(True);
  assert_array_equals(tmp6, [d2]);
  tmp7 ← n . ''s3'';
  tmp8 ← tmp7 . assignedNodes(True);
  assert_array_equals(tmp8, [d2]);
```

```
  tmp9 ← n . ''s4'';
  tmp10 ← tmp9 . assignedNodes(True);
  tmp11 ← n . ''f4'';
  assert_array_equals(tmp10, [d2, tmp11])
}) slots_fallback_heap"
```
⟨*proof*⟩

'Slots fallback: Mutation. Remove an assigned node from a slot so that fallback contens will be used.'

lemma *"test (do {*
```
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''c1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedNodes(True);
  tmp7 ← n . ''f1'';
  assert_array_equals(tmp6, [tmp7]);
  tmp8 ← n . ''s2'';
  tmp9 ← tmp8 . assignedNodes(True);
  tmp10 ← n . ''f1'';
  tmp11 ← n . ''f2'';
  assert_array_equals(tmp9, [tmp10, tmp11]);
  tmp12 ← n . ''s3'';
  tmp13 ← tmp12 . assignedNodes(True);
  tmp14 ← n . ''f1'';
  tmp15 ← n . ''f2'';
  assert_array_equals(tmp13, [tmp14, tmp15]);
  tmp16 ← n . ''s4'';
  tmp17 ← tmp16 . assignedNodes(True);
  tmp18 ← n . ''f1'';
  tmp19 ← n . ''f2'';
  tmp20 ← n . ''f4'';
  assert_array_equals(tmp17, [tmp18, tmp19, tmp20])
}) slots_fallback_heap"
```
⟨*proof*⟩

'Slots fallback: Mutation. Remove a slot which is a fallback content of another slot.'

lemma *"test (do {*
```
  tmp0 ← slots_fallback_document . getElementById(''test5'');
  n ← createTestTree(tmp0);
  tmp1 ← n . ''test5'';
  removeWhiteSpaceOnlyTextNodes(tmp1);
  tmp2 ← n . ''s1'';
  tmp2 . remove();
  tmp3 ← n . ''s1'';
  tmp4 ← tmp3 . assignedNodes();
  assert_array_equals(tmp4, []);
  tmp5 ← n . ''s1'';
  tmp6 ← tmp5 . assignedNodes(True);
  assert_array_equals(tmp6, [], ''fall back contents should be empty because s1 is not in a shadow tree.'');
  tmp7 ← n . ''s2'';
  tmp8 ← tmp7 . assignedNodes(True);
  tmp9 ← n . ''f2'';
  assert_array_equals(tmp8, [tmp9]);
  tmp10 ← n . ''s3'';
  tmp11 ← tmp10 . assignedNodes(True);
  tmp12 ← n . ''f2'';
  assert_array_equals(tmp11, [tmp12]);
```

```
  tmp13 ← n . ''s4'';
  tmp14 ← tmp13 . assignedNodes(True);
  tmp15 ← n . ''f2'';
  tmp16 ← n . ''f4'';
  assert_array_equals(tmp14, [tmp15, tmp16])
}) slots_fallback_heap"
```
⟨*proof*⟩

end

3.4 Testing Document_adoptNode (Shadow_DOM_Document_adoptNode)

This theory contains the test cases for Document_adoptNode.

theory *Shadow_DOM_Document_adoptNode*
imports
"Shadow_DOM_BaseTest"
begin

definition *Document_adoptNode_heap :: heap*$_{final}$ **where**
```
  "Document_adoptNode_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
8)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''meta'' [] (fmap_of_list [(''charset'', ''utf-8'')])
None)),
    (cast (element_ptr.Ref 4), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Document.adoptNode'')),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''link'' [] (fmap_of_list [(''rel'', ''help''),
(''href'', ''https://dom.spec.whatwg.org/#dom-document-adoptnode'')]) None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''body'' [cast (element_ptr.Ref 9), cast (element_ptr.Ref
10), cast (element_ptr.Ref 11)] fmempty None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')]) None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''x<'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''x'')),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

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

"Adopting an Element called 'x<' should work."

lemma *"test (do {*
```
  tmp0 ← Document_adoptNode_document . getElementsByTagName(''x<'');
  y ← return (tmp0 ! 0);
  child ← y . firstChild;
  tmp1 ← y . parentNode;
  tmp2 ← Document_adoptNode_document . body;
  assert_equals(tmp1, tmp2);
  tmp3 ← y . ownerDocument;
  assert_equals(tmp3, Document_adoptNode_document);
```

```
  tmp4 ← Document_adoptNode_document . adoptNode(y);
  assert_equals(tmp4, y);
  tmp5 ← y . parentNode;
  assert_equals(tmp5, None);
  tmp6 ← y . firstChild;
  assert_equals(tmp6, child);
  tmp7 ← y . ownerDocument;
  assert_equals(tmp7, Document_adoptNode_document);
  tmp8 ← child . ownerDocument;
  assert_equals(tmp8, Document_adoptNode_document);
  doc ← createDocument(None, None, None);
  tmp9 ← doc . adoptNode(y);
  assert_equals(tmp9, y);
  tmp10 ← y . parentNode;
  assert_equals(tmp10, None);
  tmp11 ← y . firstChild;
  assert_equals(tmp11, child);
  tmp12 ← y . ownerDocument;
  assert_equals(tmp12, doc);
  tmp13 ← child . ownerDocument;
  assert_equals(tmp13, doc)
}) Document_adoptNode_heap"
```

⟨*proof*⟩

"Adopting an Element called ':good:times:' should work."

lemma *"test (do {*

```
  x ← Document_adoptNode_document . createElement('':good:times:'');
  tmp0 ← Document_adoptNode_document . adoptNode(x);
  assert_equals(tmp0, x);
  doc ← createDocument(None, None, None);
  tmp1 ← doc . adoptNode(x);
  assert_equals(tmp1, x);
  tmp2 ← x . parentNode;
  assert_equals(tmp2, None);
  tmp3 ← x . ownerDocument;
  assert_equals(tmp3, doc)
}) Document_adoptNode_heap"
```

⟨*proof*⟩

end

3.5 Testing Document_getElementById (Shadow_DOM_Document_getElementById)

This theory contains the test cases for Document_getElementById.

theory *Shadow_DOM_Document_getElementById*
imports
"Shadow_DOM_BaseTest"
begin

definition *Document_getElementById_heap :: heap*$_{final}$ **where**

```
  "Document_getElementById_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
9)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6), cast (element_ptr.Ref 7), cast (element_ptr.Ref
8)] fmempty None)),
    (cast (element_ptr.Ref 3), cast (create_element_obj ''meta'' [] (fmap_of_list [(''charset'', ''utf-8'')])
None)),
```

```
    (cast (element_ptr.Ref 4), cast (create_element_obj ''title'' [cast (character_data_ptr.Ref 1)] fmempty
None)),
    (cast (character_data_ptr.Ref 1), cast (create_character_data_obj ''Document.getElementById'')),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''link'' [] (fmap_of_list [(''rel'', ''author''),
(''title'', ''Tetsuharu OHZEKI''), (''href'', ''mailto:saneyuki.snyk@gmail.com'')]) None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''link'' [] (fmap_of_list [(''rel'', ''help''),
(''href'', ''https://dom.spec.whatwg.org/#dom-document-getelementbyid'')]) None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''body'' [cast (element_ptr.Ref 10), cast (element_ptr.Ref
11), cast (element_ptr.Ref 12), cast (element_ptr.Ref 13), cast (element_ptr.Ref 16), cast (element_ptr.Ref
19)] fmempty None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')]) None)),
    (cast (element_ptr.Ref 11), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', '''')]) None)),
    (cast (element_ptr.Ref 12), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''test1'')])
None)),
    (cast (element_ptr.Ref 13), cast (create_element_obj ''div'' [cast (element_ptr.Ref 14), cast (element_ptr.Ref
15)] (fmap_of_list [(''id'', ''test5''), (''data-name'', ''1st'')]) None)),
    (cast (element_ptr.Ref 14), cast (create_element_obj ''p'' [cast (character_data_ptr.Ref 2)] (fmap_of_list
[(''id'', ''test5''), (''data-name'', ''2nd'')]) None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''P'')),
    (cast (element_ptr.Ref 15), cast (create_element_obj ''input'' [] (fmap_of_list [(''id'', ''test5''),
(''type'', ''submit''), (''value'', ''Submit''), (''data-name'', ''3rd'')]) None)),
    (cast (element_ptr.Ref 16), cast (create_element_obj ''div'' [cast (element_ptr.Ref 17)] (fmap_of_list
[(''id'', ''outer'')]) None)),
    (cast (element_ptr.Ref 17), cast (create_element_obj ''div'' [cast (element_ptr.Ref 18)] (fmap_of_list
[(''id'', ''middle'')]) None)),
    (cast (element_ptr.Ref 18), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''inner'')])
None)),
    (cast (element_ptr.Ref 19), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 3)] fmempty
None)),
    (cast (character_data_ptr.Ref 3), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

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

"Document.getElementById with a script-inserted element"

lemma *"test (do {*
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test2'';
  test ← Document_getElementById_document . createElement(''div'');
  test . setAttribute(''id'', TEST_ID);
  gBody . appendChild(test);
  result ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(result, None, ''should not be null.'');
  tmp0 ← result . tagName;
  assert_equals(tmp0, ''div'', ''should have appended element's tag name'');
  gBody . removeChild(test);
  removed ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(removed, None, ''should not get removed element.'')
}) Document_getElementById_heap"

⟨*proof*⟩

"update ‘id‘ attribute via setAttribute/removeAttribute"

lemma *"test (do {*
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test3'';
  test ← Document_getElementById_document . createElement(''div'');
  test . setAttribute(''id'', TEST_ID);
  gBody . appendChild(test);
  UPDATED_ID ← return ''test3-updated'';

```
  test . setAttribute(''id'', UPDATED_ID);
  e ← Document_getElementById_document . getElementById(UPDATED_ID);
  assert_equals(e, test, ''should get the element with id.'');
  old ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(old, None, ''shouldn't get the element by the old id.'');
  test . removeAttribute(''id'');
  e2 ← Document_getElementById_document . getElementById(UPDATED_ID);
  assert_equals(e2, None, ''should return null when the passed id is none in document.'')
}) Document_getElementById_heap"
```
⟨*proof*⟩

"Ensure that the id attribute only affects elements present in a document"

lemma *"test (do {*
```
  TEST_ID ← return ''test4-should-not-exist'';
  e ← Document_getElementById_document . createElement(''div'');
  e . setAttribute(''id'', TEST_ID);
  tmp0 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp0, None, ''should be null'');
  tmp1 ← Document_getElementById_document . body;
  tmp1 . appendChild(e);
  tmp2 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp2, e, ''should be the appended element'')
}) Document_getElementById_heap"
```
⟨*proof*⟩

"in tree order, within the context object's tree"

lemma *"test (do {*
```
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test5'';
  target ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(target, None, ''should not be null'');
  tmp0 ← target . getAttribute(''data-name'');
  assert_equals(tmp0, ''1st'', ''should return the 1st'');
  element4 ← Document_getElementById_document . createElement(''div'');
  element4 . setAttribute(''id'', TEST_ID);
  element4 . setAttribute(''data-name'', ''4th'');
  gBody . appendChild(element4);
  target2 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(target2, None, ''should not be null'');
  tmp1 ← target2 . getAttribute(''data-name'');
  assert_equals(tmp1, ''1st'', ''should be the 1st'');
  tmp2 ← target2 . parentNode;
  tmp2 . removeChild(target2);
  target3 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_not_equals(target3, None, ''should not be null'');
  tmp3 ← target3 . getAttribute(''data-name'');
  assert_equals(tmp3, ''4th'', ''should be the 4th'')
}) Document_getElementById_heap"
```
⟨*proof*⟩

"Modern browsers optimize this method with using internal id cache. This test checks that their optimization should effect only append to ‘Document‘, not append to ‘Node‘."

lemma *"test (do {*
```
  TEST_ID ← return ''test6'';
  s ← Document_getElementById_document . createElement(''div'');
  s . setAttribute(''id'', TEST_ID);
  tmp0 ← Document_getElementById_document . createElement(''div'');
  tmp0 . appendChild(s);
  tmp1 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp1, None, ''should be null'')
}) Document_getElementById_heap"
```
⟨*proof*⟩

"changing attribute's value via ‘Attr‘ gotten from ‘Element.attribute‘."

lemma *"test (do {*
gBody ← Document_getElementById_document . body;
TEST_ID ← return ''test7'';
element ← Document_getElementById_document . createElement(''div'');
element . setAttribute(''id'', TEST_ID);
gBody . appendChild(element);
target ← Document_getElementById_document . getElementById(TEST_ID);
assert_equals(target, element, ''should return the element before changing the value'');
element . setAttribute(''id'', (TEST_ID @ ''-updated''));
target2 ← Document_getElementById_document . getElementById(TEST_ID);
assert_equals(target2, None, ''should return null after updated id via Attr.value'');
target3 ← Document_getElementById_document . getElementById((TEST_ID @ ''-updated''));
assert_equals(target3, element, ''should be equal to the updated element.'')
}) Document_getElementById_heap"
⟨*proof*⟩

"update ‘id‘ attribute via element.id"

lemma *"test (do {*
gBody ← Document_getElementById_document . body;
TEST_ID ← return ''test12'';
test ← Document_getElementById_document . createElement(''div'');
test . setAttribute(''id'', TEST_ID);
gBody . appendChild(test);
UPDATED_ID ← return (TEST_ID @ ''-updated'');
test . setAttribute(''id'', UPDATED_ID);
e ← Document_getElementById_document . getElementById(UPDATED_ID);
assert_equals(e, test, ''should get the element with id.'');
old ← Document_getElementById_document . getElementById(TEST_ID);
assert_equals(old, None, ''shouldn't get the element by the old id.'');
test . setAttribute(''id'', '''');
e2 ← Document_getElementById_document . getElementById(UPDATED_ID);
assert_equals(e2, None, ''should return null when the passed id is none in document.'')
}) Document_getElementById_heap"
⟨*proof*⟩

"where insertion order and tree order don't match"

lemma *"test (do {*
gBody ← Document_getElementById_document . body;
TEST_ID ← return ''test13'';
container ← Document_getElementById_document . createElement(''div'');
container . setAttribute(''id'', (TEST_ID @ ''-fixture''));
gBody . appendChild(container);
element1 ← Document_getElementById_document . createElement(''div'');
element1 . setAttribute(''id'', TEST_ID);
element2 ← Document_getElementById_document . createElement(''div'');
element2 . setAttribute(''id'', TEST_ID);
element3 ← Document_getElementById_document . createElement(''div'');
element3 . setAttribute(''id'', TEST_ID);
element4 ← Document_getElementById_document . createElement(''div'');
element4 . setAttribute(''id'', TEST_ID);
container . appendChild(element2);
container . appendChild(element4);
container . insertBefore(element3, element4);
container . insertBefore(element1, element2);
test ← Document_getElementById_document . getElementById(TEST_ID);
assert_equals(test, element1, ''should return 1st element'');
container . removeChild(element1);
test ← Document_getElementById_document . getElementById(TEST_ID);
assert_equals(test, element2, ''should return 2nd element'');
container . removeChild(element2);
test ← Document_getElementById_document . getElementById(TEST_ID);

```
  assert_equals(test, element3, ''should return 3rd element'');
  container . removeChild(element3);
  test ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(test, element4, ''should return 4th element'');
  container . removeChild(element4)
}) Document_getElementById_heap"
```
⟨*proof*⟩

"Inserting an id by inserting its parent node"

lemma *"test (do {*
```
  gBody ← Document_getElementById_document . body;
  TEST_ID ← return ''test14'';
  a ← Document_getElementById_document . createElement(''a'');
  b ← Document_getElementById_document . createElement(''b'');
  a . appendChild(b);
  b . setAttribute(''id'', TEST_ID);
  tmp0 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp0, None);
  gBody . appendChild(a);
  tmp1 ← Document_getElementById_document . getElementById(TEST_ID);
  assert_equals(tmp1, b)
}) Document_getElementById_heap"
```
⟨*proof*⟩

"Document.getElementById must not return nodes not present in document"

lemma *"test (do {*
```
  TEST_ID ← return ''test15'';
  outer ← Document_getElementById_document . getElementById(''outer'');
  middle ← Document_getElementById_document . getElementById(''middle'');
  inner ← Document_getElementById_document . getElementById(''inner'');
  tmp0 ← Document_getElementById_document . getElementById(''middle'');
  outer . removeChild(tmp0);
  new_el ← Document_getElementById_document . createElement(''h1'');
  new_el . setAttribute(''id'', ''heading'');
  inner . appendChild(new_el);
  tmp1 ← Document_getElementById_document . getElementById(''heading'');
  assert_equals(tmp1, None)
}) Document_getElementById_heap"
```
⟨*proof*⟩

end

3.6 Testing Node_insertBefore (Shadow_DOM_Node_insertBefore)

This theory contains the test cases for Node_insertBefore.

theory *Shadow_DOM_Node_insertBefore*
imports
```
  "Shadow_DOM_BaseTest"
```
begin

definition *Node_insertBefore_heap :: heap*$_{final}$ **where**
```
  "Node_insertBefore_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
6)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5)] 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 ''Node.insertBefore'')),
```

```
    (cast (element_ptr.Ref 4), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''body'' [cast (element_ptr.Ref 7), cast (element_ptr.Ref
8)] fmempty None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')]) None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

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

"Calling insertBefore an a leaf node Text must throw HIERARCHY_REQUEST_ERR."

lemma *"test (do {*
* node ← Node_insertBefore_document . createTextNode(''Foo'');*
* tmp0 ← Node_insertBefore_document . createTextNode(''fail'');*
* assert_throws(HierarchyRequestError, node . insertBefore(tmp0, None))*
}) Node_insertBefore_heap"
⟨*proof*⟩

"Calling insertBefore with an inclusive ancestor of the context object must throw HIERARCHY_REQUEST_ERR."

lemma *"test (do {*
* tmp1 ← Node_insertBefore_document . body;*
* tmp2 ← Node_insertBefore_document . getElementById(''log'');*
* tmp0 ← Node_insertBefore_document . body;*
* assert_throws(HierarchyRequestError, tmp0 . insertBefore(tmp1, tmp2));*
* tmp4 ← Node_insertBefore_document . documentElement;*
* tmp5 ← Node_insertBefore_document . getElementById(''log'');*
* tmp3 ← Node_insertBefore_document . body;*
* assert_throws(HierarchyRequestError, tmp3 . insertBefore(tmp4, tmp5))*
}) Node_insertBefore_heap"
⟨*proof*⟩

"Calling insertBefore with a reference child whose parent is not the context node must throw a NotFoundError."

lemma *"test (do {*
* a ← Node_insertBefore_document . createElement(''div'');*
* b ← Node_insertBefore_document . createElement(''div'');*
* c ← Node_insertBefore_document . createElement(''div'');*
* assert_throws(NotFoundError, a . insertBefore(b, c))*
}) Node_insertBefore_heap"
⟨*proof*⟩

"If the context node is a document, inserting a document or text node should throw a HierarchyRequestError."

lemma *"test (do {*
* doc ← createDocument(''title'');*
* doc2 ← createDocument(''title2'');*
* tmp0 ← doc . documentElement;*
* assert_throws(HierarchyRequestError, doc . insertBefore(doc2, tmp0));*
* tmp1 ← doc . createTextNode(''text'');*
* tmp2 ← doc . documentElement;*
* assert_throws(HierarchyRequestError, doc . insertBefore(tmp1, tmp2))*
}) Node_insertBefore_heap"
⟨*proof*⟩

"Inserting a node before itself should not move the node"

lemma *"test (do {*
* a ← Node_insertBefore_document . createElement(''div'');*
* b ← Node_insertBefore_document . createElement(''div'');*
* c ← Node_insertBefore_document . createElement(''div'');*
* a . appendChild(b);*

```
  a . appendChild(c);
  tmp0 ← a . childNodes;
  assert_array_equals(tmp0, [b, c]);
  tmp1 ← a . insertBefore(b, b);
  assert_equals(tmp1, b);
  tmp2 ← a . childNodes;
  assert_array_equals(tmp2, [b, c]);
  tmp3 ← a . insertBefore(c, c);
  assert_equals(tmp3, c);
  tmp4 ← a . childNodes;
  assert_array_equals(tmp4, [b, c])
}) Node_insertBefore_heap"
```
⟨*proof*⟩

end

3.7 Testing Node_removeChild (Shadow_DOM_Node_removeChild)

This theory contains the test cases for Node_removeChild.

theory *Shadow_DOM_Node_removeChild*
imports
"Shadow_DOM_BaseTest"
begin

definition *Node_removeChild_heap :: heap*$_{final}$ **where**
```
  "Node_removeChild_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
7)] fmempty None)),
    (cast (element_ptr.Ref 2), cast (create_element_obj ''head'' [cast (element_ptr.Ref 3), cast (element_ptr.Ref
4), cast (element_ptr.Ref 5), cast (element_ptr.Ref 6)] 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 ''Node.removeChild'')),
    (cast (element_ptr.Ref 4), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 5), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''/resources/testhar
None)),
    (cast (element_ptr.Ref 6), cast (create_element_obj ''script'' [] (fmap_of_list [(''src'', ''creators.js'')])
None)),
    (cast (element_ptr.Ref 7), cast (create_element_obj ''body'' [cast (element_ptr.Ref 8), cast (element_ptr.Ref
9), cast (element_ptr.Ref 10)] fmempty None)),
    (cast (element_ptr.Ref 8), cast (create_element_obj ''div'' [] (fmap_of_list [(''id'', ''log'')])) None)),
    (cast (element_ptr.Ref 9), cast (create_element_obj ''iframe'' [] (fmap_of_list [(''src'', ''about:blank'')])
None)),
    (cast (element_ptr.Ref 10), cast (create_element_obj ''script'' [cast (character_data_ptr.Ref 2)] fmempty
None)),
    (cast (character_data_ptr.Ref 2), cast (create_character_data_obj ''%3C%3Cscript%3E%3E''))]"
```

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

"Passing a detached Element to removeChild should not affect it."

lemma *"test (do {*
```
  doc ← return Node_removeChild_document;
  s ← doc . createElement(''div'');
  tmp0 ← s . ownerDocument;
  assert_equals(tmp0, doc);
  tmp1 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp1 . removeChild(s));
  tmp2 ← s . ownerDocument;
```

```
  assert_equals(tmp2, doc)
}) Node_removeChild_heap"
```
⟨*proof*⟩

"Passing a non-detached Element to removeChild should not affect it."

lemma *"test (do {*
```
  doc ← return Node_removeChild_document;
  s ← doc . createElement(''div'');
  tmp0 ← doc . documentElement;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  tmp2 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp2 . removeChild(s));
  tmp3 ← s . ownerDocument;
  assert_equals(tmp3, doc)
}) Node_removeChild_heap"
```
⟨*proof*⟩

"Calling removeChild on an Element with no children should throw NOT_FOUND_ERR."

lemma *"test (do {*
```
  doc ← return Node_removeChild_document;
  s ← doc . createElement(''div'');
  tmp0 ← doc . body;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  assert_throws(NotFoundError, s . removeChild(doc))
}) Node_removeChild_heap"
```
⟨*proof*⟩

"Passing a detached Element to removeChild should not affect it."

lemma *"test (do {*
```
  doc ← createDocument('''');
  s ← doc . createElement(''div'');
  tmp0 ← s . ownerDocument;
  assert_equals(tmp0, doc);
  tmp1 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp1 . removeChild(s));
  tmp2 ← s . ownerDocument;
  assert_equals(tmp2, doc)
}) Node_removeChild_heap"
```
⟨*proof*⟩

"Passing a non-detached Element to removeChild should not affect it."

lemma *"test (do {*
```
  doc ← createDocument('''');
  s ← doc . createElement(''div'');
  tmp0 ← doc . documentElement;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  tmp2 ← Node_removeChild_document . body;
  assert_throws(NotFoundError, tmp2 . removeChild(s));
  tmp3 ← s . ownerDocument;
  assert_equals(tmp3, doc)
}) Node_removeChild_heap"
```
⟨*proof*⟩

"Calling removeChild on an Element with no children should throw NOT_FOUND_ERR."

lemma *"test (do {*
```
  doc ← createDocument('''');
```

```
  s ← doc . createElement(''div'');
  tmp0 ← doc . body;
  tmp0 . appendChild(s);
  tmp1 ← s . ownerDocument;
  assert_equals(tmp1, doc);
  assert_throws(NotFoundError, s . removeChild(doc))
}) Node_removeChild_heap"
```
⟨*proof*⟩

"Passing a value that is not a Node reference to removeChild should throw TypeError."

lemma *"test (do {*
```
  tmp0 ← Node_removeChild_document . body;
  assert_throws(TypeError, tmp0 . removeChild(None))
}) Node_removeChild_heap"
```
⟨*proof*⟩

end

3.8 Shadow DOM Tests (Shadow_DOM_Tests)

theory *Shadow_DOM_Tests*
imports
"tests/slots"
"tests/slots_fallback"
"tests/Shadow_DOM_Document_adoptNode"
"tests/Shadow_DOM_Document_getElementById"
"tests/Shadow_DOM_Node_insertBefore"
"tests/Shadow_DOM_Node_removeChild"
begin
end

Bibliography

[1] A. Bohannon and B. C. Pierce. Featherweight Firefox: Formalizing the core of a web browser. In *Usenix Conference on Web Application Development (WebApps)*, June 2010. URL `http://www.cis.upenn.edu/~bohannon/browser-model/`.

[2] A. D. Brucker. *An Interactive Proof Environment for Object-oriented Specifications*. PhD thesis, ETH Zurich, mar 2007. URL `https://www.brucker.ch/bibliography/abstract/brucker-interactive-2007`. ETH Dissertation No. 17097.

[3] 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.

[4] A. D. Brucker and M. Herzberg. Formalizing (web) standards: An application of test and proof. In C. Dubois and B. Wolff, editors, *TAP 2018: Tests And Proofs*, number 10889 in Lecture Notes in Computer Science, pages 159–166. Springer-Verlag, Heidelberg, 2018. ISBN 978-3-642-38915-3. doi: 10.1007/978-3-319-92994-1_9. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-standard-compliance-testing-2018`.

[5] A. D. Brucker and M. Herzberg. A formalization of web components. *Archive of Formal Proofs*, Sept. 2020. ISSN 2150-914x. `http://www.isa-afp.org/entries/DOM_Components.html`, Formal proof development.

[6] 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 `https://www.brucker.ch/bibliography/abstract/brucker.ea-web-components-2019`.

[7] A. D. Brucker and B. Wolff. Interactive testing using HOL-TestGen. In W. Grieskamp and C. Weise, editors, *Formal Approaches to Testing of Software*, number 3997 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2005. ISBN 3-540-25109-X. doi: 10.1007/11759744_7. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-interactive-2005`.

[8] A. D. Brucker and B. Wolff. An extensible encoding of object-oriented data models in hol. *Journal of Automated Reasoning*, 41:219–249, 2008. ISSN 0168-7433. doi: 10.1007/s10817-008-9108-3. URL `https://www.brucker.ch/bibliography/abstract/brucker.ea-extensible-2008-b`.

[9] A. D. Brucker and B. Wolff. On theorem prover-based testing. *Formal Aspects of Computing*, 25(5):683–721, 2013. ISSN 0934-5043. doi: 10.1007/s00165-012-0222-y. URL `http://www.brucker.ch/bibliography/abstract/brucker.ea-theorem-prover-2012`.

[10] P. Gardner, G. Smith, M. J. Wheelhouse, and U. Zarfaty. DOM: towards a formal specification. In *PLAN-X 2008, Programming Language Technologies for XML, An ACM SIGPLAN Workshop colocated with POPL 2008, San Francisco, California, USA, January 9, 2008*, 2008. URL `http://gemo.futurs.inria.fr/events/PLANX2008/papers/p18.pdf`.

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

[12] D. Jang, Z. Tatlock, and S. Lerner. Establishing browser security guarantees through formal shim verification. In T. Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 113–128. USENIX Association, 2012. URL `https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/jang`.

[13] G. Klein. Operating system verification — an overview. *Sādhanā*, 34(1):27–69, Feb. 2009.

[14] A. Raad, J. F. Santos, and P. Gardner. DOM: specification and client reasoning. In A. Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 401–422, 2016. ISBN 978-3-319-47957-6. doi: 10.1007/978-3-319-47958-3_21.

[15] W3C. W3C DOM4, Nov. 2015. URL `https://www.w3.org/TR/dom/`.

[16] WHATWG. DOM – living standard, Mar. 2017. URL `https://dom.spec.whatwg.org/commit-snapshots/6253e53af2fbfaa6d25ad09fd54280d8083b2a97/`. Last Updated 24 March 2017.