

Modal Abstractions for OS Kernels

Ph.D. Defense

Ismail Kuru

Department of Computer Science
Drexel University

March 14, 2025

Overview

Part I: Modal Understanding of Location Virtualization

1. Definitions

2. Motivation

- 2.1 Sharing
- 2.2 Address-Space Agnostic Virtual Memory References

3. Logic

- 3.1 A Quick Tour on Separation Logic (**SL**)
- 3.2 A Logic for Recovering the Soundness of Page-Table Traversal
- 3.3 A Logic of Agnostic Resources

4. The Verification Effort

- 4.1 Design

5. The Current and Future Directions & Conclusions

- 5.1 Modal Abstractions for Verification Patterns

Overview

Part II: Modal Understanding of Specification Evolution

6. Definitions

7. Motivation

7.1 Indistinguishability of Machines

8. Logic

8.1 Iris **STS** Library

8.2 Understanding Bisimulations

8.3 Invariants

9. The Current and Future Directions & Conclusions

Part I

Modal Understanding of Location Virtualization

The Essentials in Systems Programming

a supposedly allocated physical resource

1

pointer va :=
a virtual reference

malloc (size)

Memory Location Virtualization

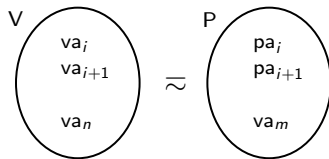


Figure: Virtualization: The Deception of Abundance

Memory Location Virtualization: Abstraction

An Address Space with Logical Name γ

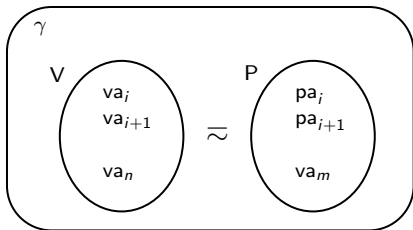


Figure: Address-Spaces: Named Containers for Virtual Memory Mappings

A Program Named γ_n

γ_n

```
pointer va :=  
  malloc(size)
```

A Program Named γ_m

γ_m

```
pointer va :=  
  malloc(size)
```

- A program is abstracted as a *named address-space*
- A container of *virtual-to-physical* memory resource mappings

Memory Location Virtualization: Mechanism

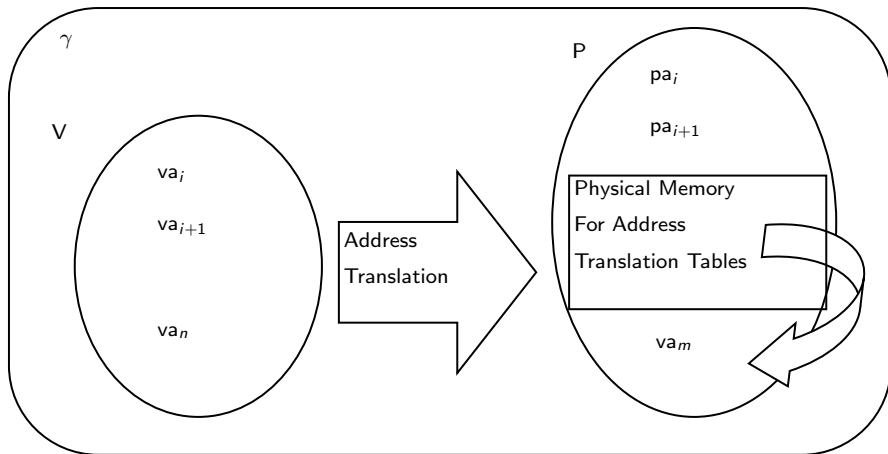


Figure: Address-Translation: A Mechanism for Realizing Address Virtualization

Page Tables

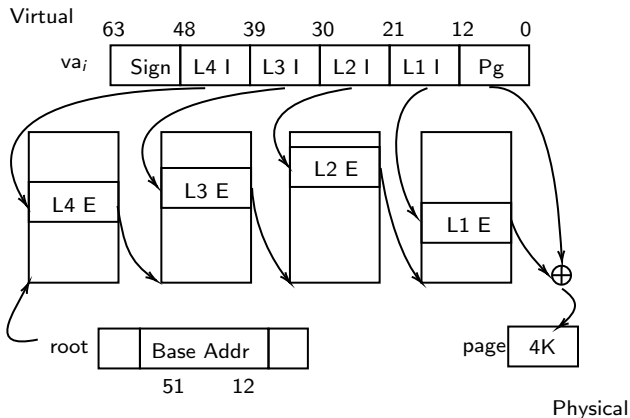
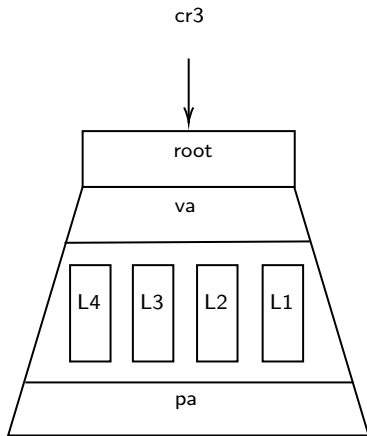


Figure: Page-Tables (**PT**): Data Structures for Address-Translation

A Complete Picture of Address-Space Abstraction



The Current View of Memory

The register `cr3` points to the current view of the memory, i.e., the loaded address space in the memory

Figure: Depicting an Address-Space with its Essential Aspects

Virtual Memory Management (VMM)

VMM as a General Resource Provider

"the virtual memory sub-system can be considered the core of a Solaris instance, and the implementation of Solaris virtual memory affects just about every other subsystem in the operating system" [McDougall and Mauro(2006)]

Sharing Physical Page Tables

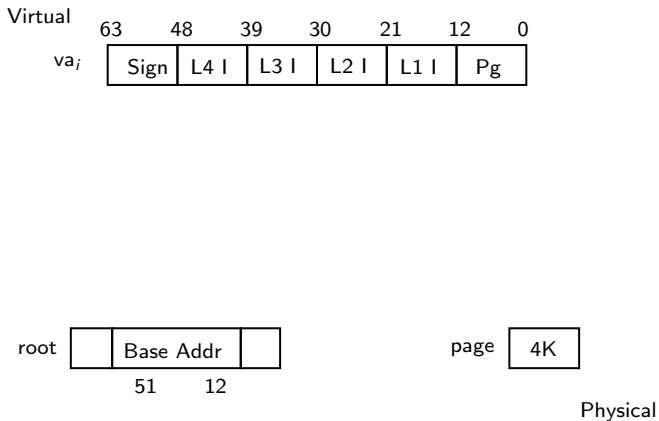
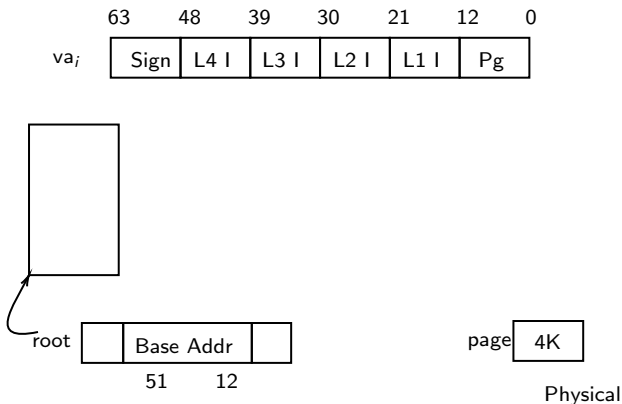


Figure: Physical and Virtual Resources

Sharing Physical Page Tables

Virtual

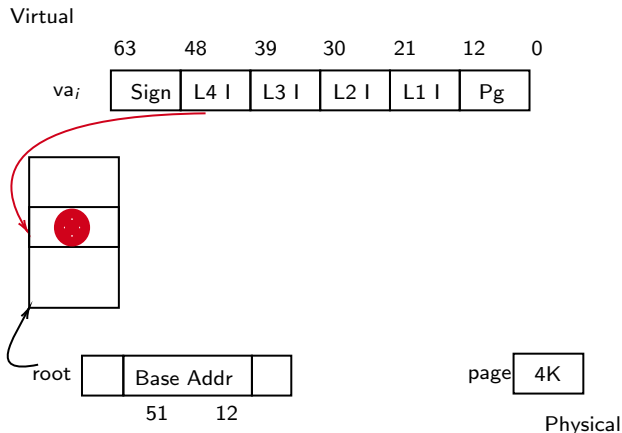


```
static pte_t *pte_next_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
```

Figure: Traversal Starts from the root

Sharing Physical Page Tables

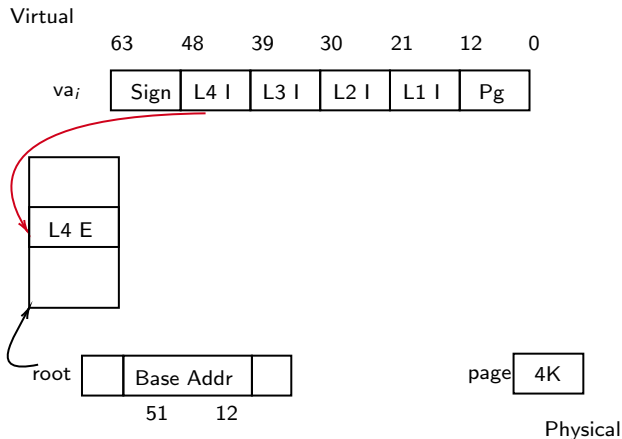


```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
}
```

Figure: Missing L4 Entry

Sharing Physical Page Tables



```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[l4l(va)];
}
```

Figure: A New L4 Entry

Sharing Physical Page Tables

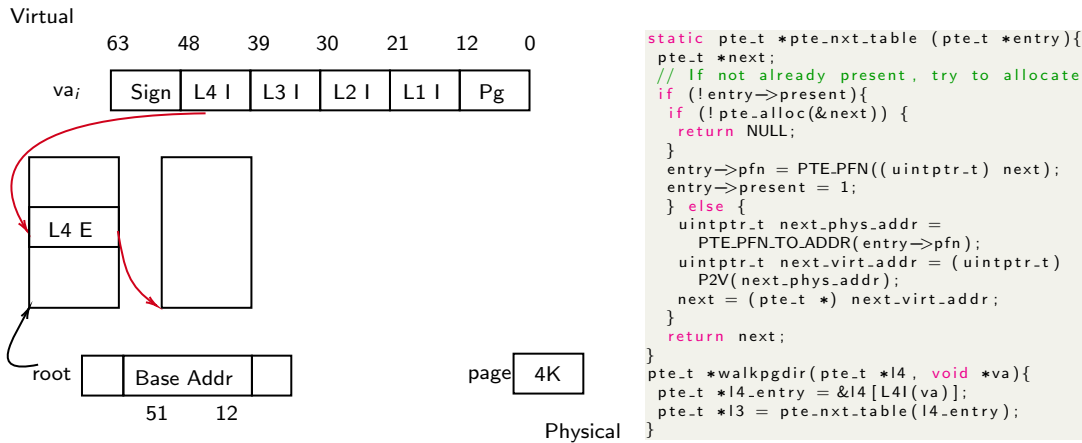
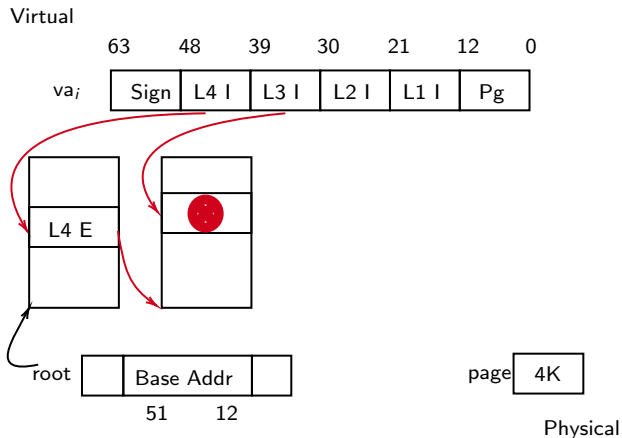


Figure: Accessing to Table L3

Sharing Physical Page Tables

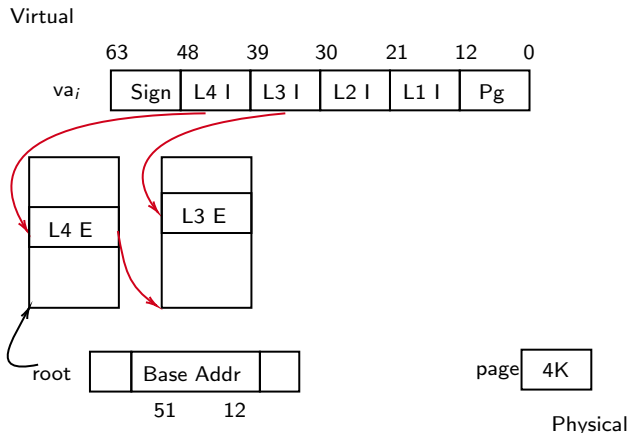


```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
}
```

Figure: Missing L3 Entry

Sharing Physical Page Tables

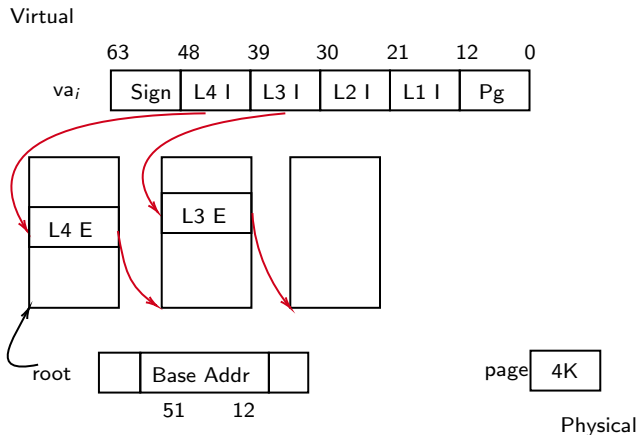


```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
}
```

Figure: A New L3 Entry

Sharing Physical Page Tables

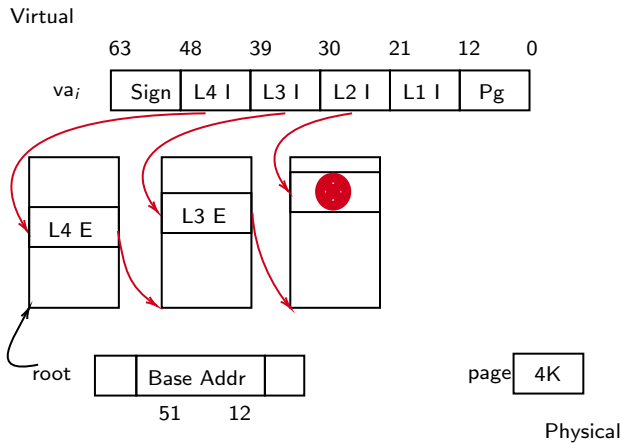


```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
    pte_t *l2 = pte_nxt_table(l3_entry);
}
```

Figure: Accessing to L2 Table

Sharing Physical Page Tables



```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
    pte_t *l2 = pte_nxt_table(l3_entry);
}
```

Figure: Missing L2 Entry

Sharing Physical Page Tables

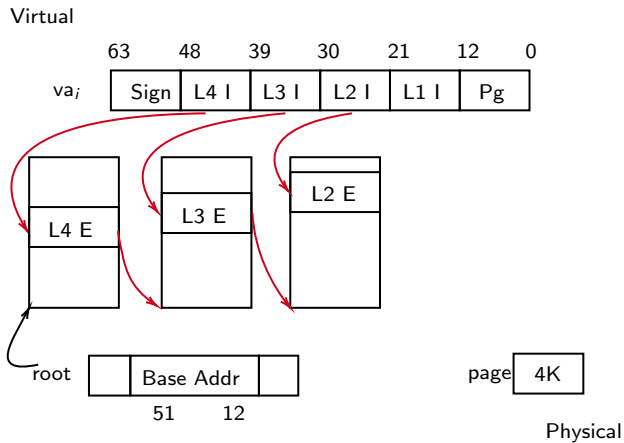


Figure: A New L2 Entry

```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
    pte_t *l2 = pte_nxt_table(l3_entry);
    pte_t *l2_entry = &l2[L2I(va)];
}
```

Sharing Physical Page Tables

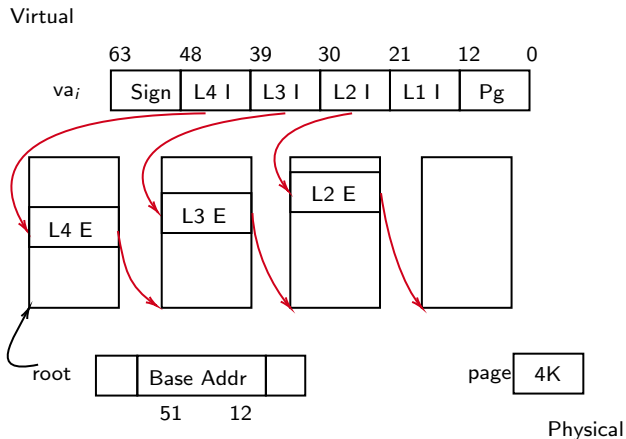


Figure: Accessing L1 Table

```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
    pte_t *l2 = pte_nxt_table(l3_entry);
    pte_t *l2_entry = &l2[L2I(va)];
    pte_t *l1 = pte_nxt_table(l2_entry);
}
```

Sharing Physical Page Tables

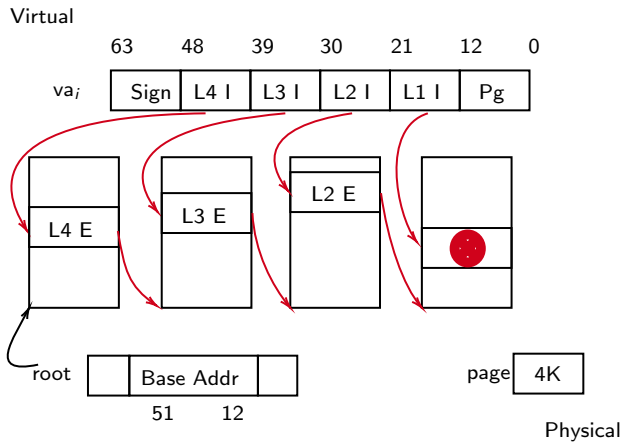
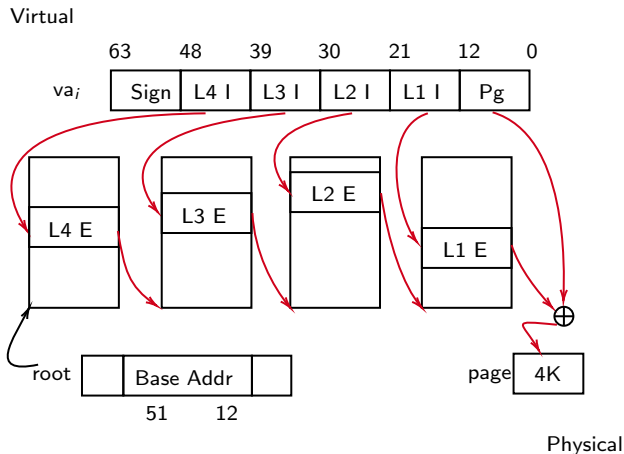


Figure: Missing L1 Entry

```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
    pte_t *l2 = pte_nxt_table(l3_entry);
    pte_t *l2_entry = &l2[L2I(va)];
    pte_t *l1 = pte_nxt_table(l2_entry);
}
```

Sharing Physical Page Tables



```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
    pte_t *l2 = pte_nxt_table(l3_entry);
    pte_t *l2_entry = &l2[L2I(va)];
    pte_t *l1 = pte_nxt_table(l2_entry);
    pte_t *l1_entry = &l1[L1I(va)];
    return l1_entry;
}
```

Figure: Accessing to the Page Referenced by L1 Entry

Breaking Soundness in Sharing

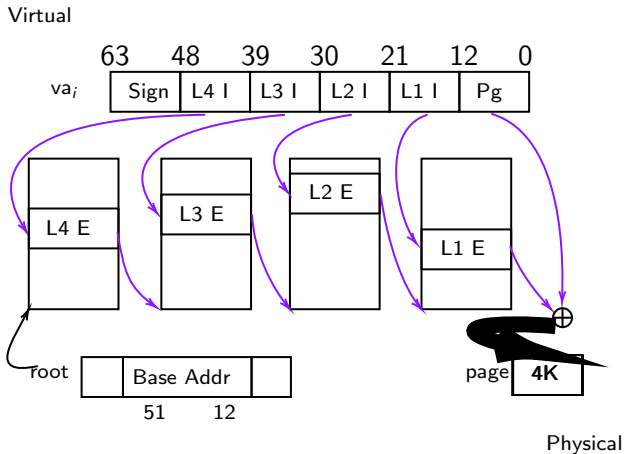
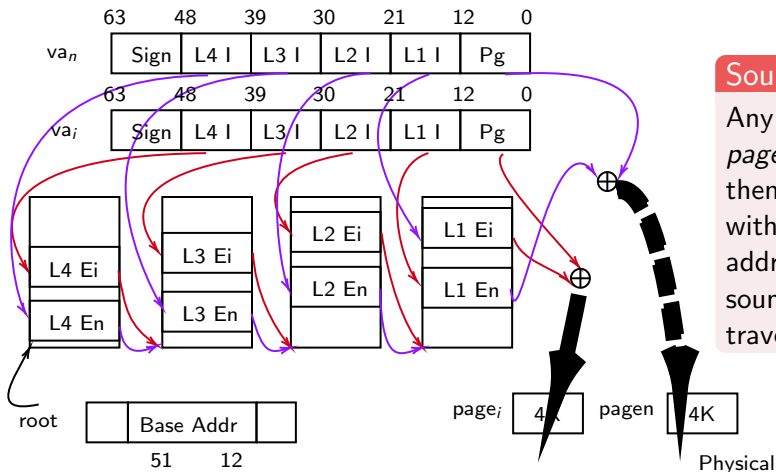


Figure: Sharing Page-Tables under Updates to Page-Tables

Breaking Soundness in Sharing

Virtual



Soundness of Traversal

Any update on the *shared page-tables*, which themselves are referenced with *physical memory* addresses, would break the soundness of any other traversal!

Managing Agnostic Memory Mappings

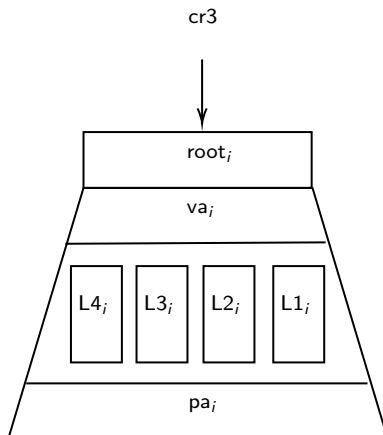


Figure: An Address Space with Unique Root Address $root_i$

Managing Agnostic Memory Mappings

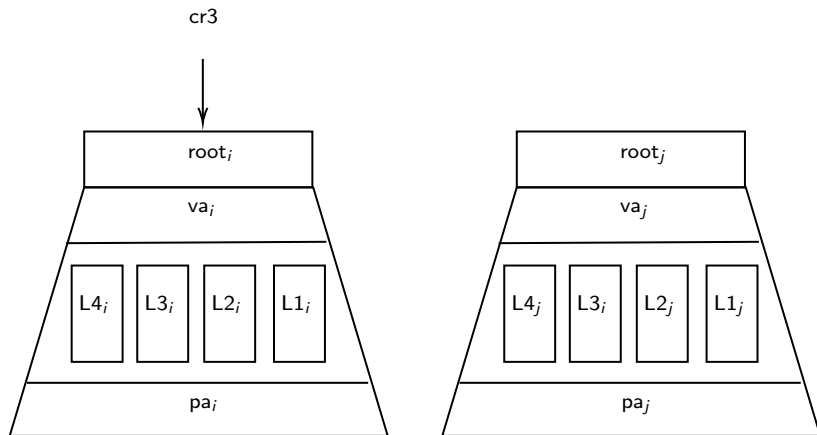


Figure: Two Address-Spaces with the Unique Root Addresses $root_i$ and $root_j$

Managing Agnostic Memory Mappings

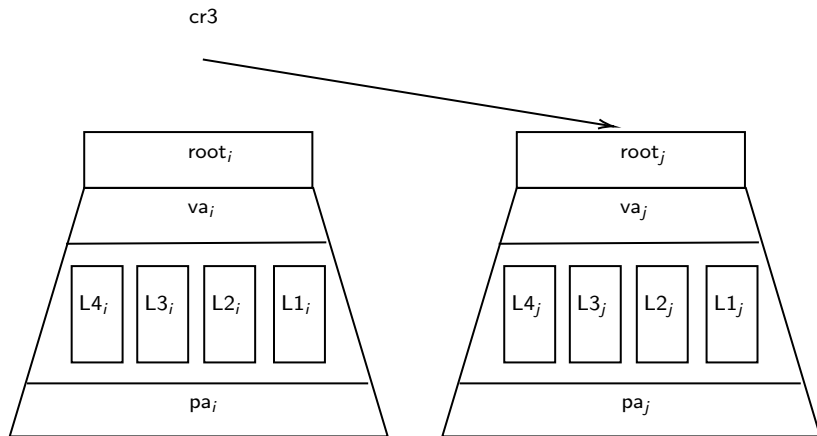


Figure: Switching Address-Spaces

Managing Agnostic Memory Mappings

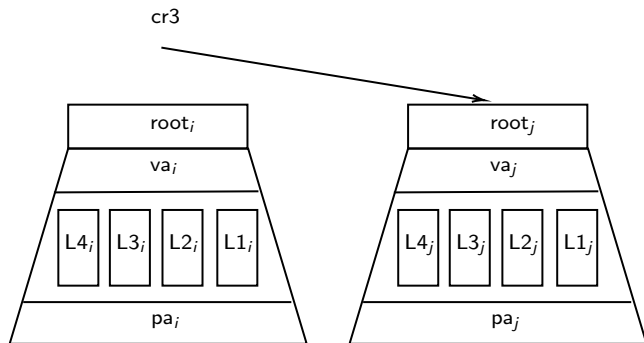


Figure: Switching Address-Spaces

Referring to Agnostic Resources

Unless we bookkeep to which address-space each of these virtual-to-physical mappings belongs, *which we never see in the practice of using virtual memory references*, we need to figure out a way of referring to these mappings as *they are only valid in their own address-spaces*.

Specifying Programs

$$\{P\} C \{Q\}$$

Separation Logic: Separating Conjunction

$$\frac{\text{FRAME} \quad \{P\} \text{ e } \{Q\}}{\{P * R\} \text{ e } \{Q * R\}}$$

Separation Logic: Ownership

- Well-known points-to assertion, e.g., $\text{memory_ref} \mapsto_q \text{val}$
- Regarding the logical machinery, Iris **SL** enables encoding a generalized form ownership of *logical resources*
- A fragmental \boxed{P}^γ ownership
 - Enabling coordinated access to logical resources
- Full \boxed{P}^γ ownership
 - Enabling access to *update* logical resources, presented as *invariants*

Separation Logic: Invariants

$$\frac{\text{INV} \quad \{P * R\} \alpha \{P * Q\}_\epsilon \quad \alpha \text{ physically atomic}}{\boxed{P}^n \vdash \{R\} \alpha \{Q\}_{\epsilon \uplus \{n\}}}$$

Defining Some Ownership Assertions

- Expected to have register ownership to be defined : $\text{reg} \mapsto_r \text{reg_val}$
- Expected to have *physical memory* ownership defined: $\text{pa} \mapsto_p \text{val}$
- How about virtual memory references?

A Naive Attempt on Virtual-Pointsto

- Page and page table addresses are *physical*
- Purple (or red) path + bold black page references are *physical*
- Why don't we define *virtual* memory references in terms of the physical page-table and the final page references?

$L_4\text{-}L_1\text{-PointsTo}(va, l4e, l3e, l2e, l1e, paddr) + paddr \mapsto_p \text{page_val}$

Tokens for Traversals

$$\underbrace{va \xrightarrow{\delta_q} pa}_{\text{Ghost translation}} * \underbrace{pa \mapsto_p \{qfrac\} val}_{\text{Physical location}}$$

- Abstract the purple and red segment of page-table traversal into *logical summarization of the walk*
- Distribute the fragmental ownership of the logical page-table summarization to virtual memory ownership

A Candidate for Kernel Invariant

$$\mathcal{IASpace}(\theta, m) \triangleq \text{ASpace_Lookup}(\theta, m) * \bigstar_{(va, paddr) \in \theta} \exists (l4e \ l3e \ l2e, l1e, paddr). L4_L1_PointsTo(va, l4e, l3e, l2e, l1e, paddr)$$

where

$$\text{ASpace_Lookup}(\theta, m) \triangleq \lambda \text{ cr3val}. \exists \delta. \vdash m !! \text{ cr3val} = \text{Some } \delta^\top * \mathcal{AbsPTableWalk}(\delta, \theta)$$

Figure: Global Address-Space Invariant with a fixed global map of address-space names m

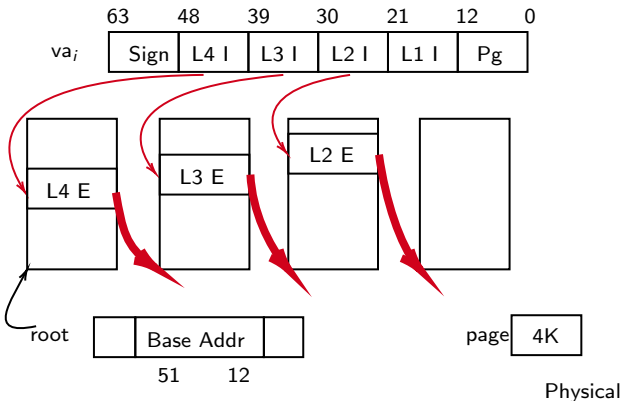
A Candidate for Kernel Invariant

The Kernel Invariant in Action

How useful (complete) is this invariant?

Kernels Locating L1 Entries

Virtual



```
static pte_t *pte_nxt_table (pte_t *entry){
    pte_t *next;
    // If not already present, try to allocate
    if (!entry->present){
        if (!pte_alloc(&next)) {
            return NULL;
        }
        entry->pfn = PTE_PFN((uintptr_t) next);
        entry->present = 1;
    } else {
        \uintptr_t next_phys_addr =
            PTE_PFN_TO_ADDR(entry->pfn);
        uintptr_t next_virt_addr = (uintptr_t)
            P2V(next_phys_addr);
        next = (pte_t *) next_virt_addr;
    }
    return next;
}

pte_t *walkpgdir(pte_t *l4, void *va){
    pte_t *l4_entry = &l4[L4I(va)];
    pte_t *l3 = pte_nxt_table(l4_entry);
    pte_t *l3_entry = &l3[L3I(va)];
    pte_t *l2 = pte_nxt_table(l3_entry);
    pte_t *l2_entry = &l2[L2I(va)];
    pte_t *l1 = pte_nxt_table(l2_entry);
    pte_t *l1_entry = &l1[L1I(va)];
}
```


Incorporating P2V into the Kernel Invariant

Definition (The Kernel Invariant for Page-Table Traversal with Virtual Page-Table Pointers)

$$\begin{aligned}
 \mathcal{I}ASpace_{id}(\theta, \Xi, m) &\triangleq ASpace_Lookup_{id}(\theta, \Xi, m) * GhostMap(id, \Xi) * \\
 &\left(\begin{aligned} &* \quad \exists (l4e, l3e, l2e, l1e, paddr). L4_L1_PointsTo(va, l4e, l3e, l2e, l1e, paddr) \end{aligned} \right) * \\
 &* \quad \exists (qfrac, q, val, va). \ulcorner va = pa + KERNBASE \text{ level} > 1 \urcorner * \underbrace{va \xrightarrow{\delta_q} pa}_{\text{Ghost translation}} * \underbrace{pa \mapsto_p \{qfrac\} val}_{\text{Physical location}} * \\
 &\quad \underbrace{\ulcorner qfrac = 1 \leftrightarrow \neg \text{entry_present}(val) \urcorner}_{\text{Entry validity}} * \\
 &\quad \underbrace{\left(\ulcorner \text{present_L}(val, level) \urcorner \multimap \forall i \in 0..511. ((\text{entry_page } val) + i * 8) \xrightarrow{id} \text{level-1} \right)}_{\text{Indexing into next level of tables}}
 \end{aligned}$$

where

$$\text{present_L}(val, level) \triangleq \text{entry_present}(val) \wedge \text{level} > 0$$

Specifying P2V

```
{ P * IASpaceid(θ, Ξ \ {entry}), m) * rbp-8 ↦v entry * rcx ↦r _ * entry ↦id _ * rtv ↦δs δ }rtv  
{ entry + KERNBASE ↦vpte,qfrac (pte_initialized (entry_val.pfn))⌈ }rtv  
{ rbp-16 ↦v (pte_initialized (entry_val.pfn)) * rax ↦r table_root (pte_initialize(entry_val.pfn)) }rtv  
{ ∀i ∈ 0 ... 511. ((table_root (pte_initialized (entry_val.pfn)))) + i * 8 ↦id v-1 }  
;; uintptr_t next_virt_addr = (uintptr_t) P2V(entry.pfn << 12);  
movabs KERNBASE, rcx { ... * rcx ↦r KERNBASE * ... }rtv  
add rcx, rax  
{ ... * rax ↦r table_root (pte_initialize(entry_val.pfn)) + KERNBASE * ... }rtv  
... ;; clean up the stack and return the rax value
```

Figure: Converting a physical address of a PTE to a virtual address (w/o instruction pointer or flag updates).

The Current Status of Machinery

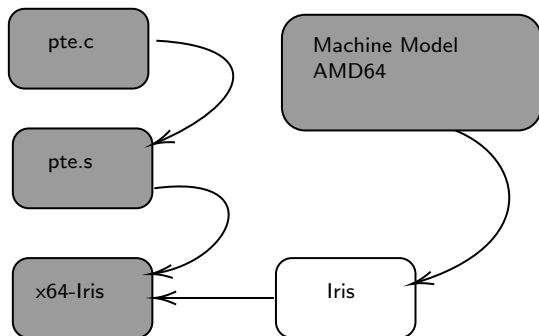


Figure: x64-Iris

- Dumping **.o** files
- Manuel treatment on **Xabs** instructions and field access

A Rough Quantification on the Current Status

Table: Line-of-Code Numbers for pte Verification

	C LoC A	Assembly LoC	Roqc Proof LoC
pte_get_next_table	12	45	3200
pte_walkpgdir	8	44	3200
pte_p2v	–	1	75
pte_switch_addrspace	–	18	350
pte_map_page	7	28	1750
pte_initialize	4	20	700

Table: Line-of-Code Numbers for x64-Iris Logic

	Roqc LoC
Soundness of Instructions Mentioned in the Thesis	50176
VMM Related Logical Constructions	5554
Machine Model	6172

Modal Abstractions as Verification Patterns in Practice

	Resource Context	Resource Elements	Nominalization	Resource Context Steps
Post-Crash Modality [Chajed(2022), Tej Chajed and contributors(2023), Chajed et al.(2019)]	$\Diamond P$	$\ell \mapsto_{\vec{n}} v$	Strong	Crash Recovery
NextGen Modality [Vindum et al.(2025)]	$\xrightarrow{t} P$	Own (t(a))	Strong	Determined Based on the Model*
StackRegion Modality* [Vindum et al.(2025)]	$\xrightarrow{Cut^n} P$	$\boxed{n} \ell \mapsto v$	Strong	Alloc and Return to/from stack
Memory-Fence Modality [Doko and Vafeiadis(2016), Doko and Vafeiadis(2017), Dang et al.(2019)]	Δ_{π} and ∇_{π}	$\ell \mapsto v$	Weak	Fence Acquire and Release
Address-Space Modality [Kuru and Gordon(2024)]	$[r]P$	$\ell \mapsto v$	Weak	Address-Space Switch
Ref-Count Modality [Wagner et al.(2024)]	$@_{\ell} P$	$\ell_1 \mapsto v$	Weak	Allocating, Dropping and Sharing a Reference

- The StackRegion Modality is an instance of NextGen (called the Independence Modality in [Vindum et al.(2025)]).

Part II

Modal Understanding of Specification Evolution

Specifying Protocols for Systems with STSes

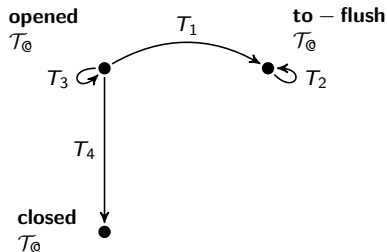


Figure: STS for Distributed File Protocol

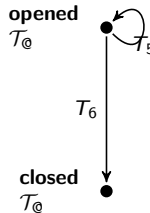


Figure: STS for Traditional File Protocol

Interacting with STSes

Modelling interactions of a client with a state machine via *token exchange*

Defining STSes

Definition (**STS** Definition following CaReSL's presentation [Turon et al.(2013)])

An STS π is given by:

1. a set of states \mathcal{S} ,
2. a map from a state set of tokens $\mathcal{T} : \mathcal{S} \rightarrow \text{TokSet}$,
3. a transition relation \rightsquigarrow on states, which is then lifted to pairs of a state and token set:

$$(s; T) \rightsquigarrow (s'; T') \triangleq s \rightsquigarrow s' \wedge \mathcal{T}(s) \uplus T = \mathcal{T}(s') \uplus T'$$

4. an interpretation mapping states to state assertions $\varphi : \mathcal{S} \rightarrow \text{Prop}$.

Propositional Kripke Model

Definition ((Propositional) Kripke Model [Hughes and Cresswell(1996)])

A Kripke model \mathfrak{M} is a triple (W, R, V) where

- W is a set of “worlds”
- $R \subseteq W \times W$ is a relation called the *accessibility* relation between worlds
- $V : \text{PropVar} \rightarrow \mathcal{P}(W)$ gives for each propositional variable p a set of worlds $V(p)$ where p is considered true

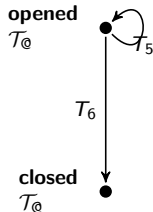
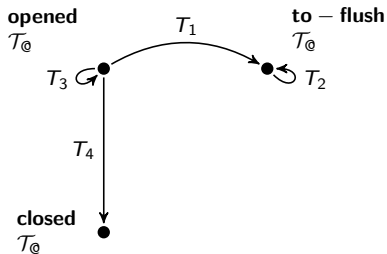
Bisimulations over Kripke Models

Definition ((Propositional) Bisimulation of Kripke Structures: $\mathfrak{M} \sim \mathfrak{M}'$.)

A *bisimulation* between (multimodal) Kripke structures $(W, R_{i \in I}, V)$ and $(W', R'_{i \in I}, V')$ is a relation $E \subseteq W \times W'$ satisfying:

- If $w E w'$, then w and w' satisfy the same propositional variables.
- If $w E w'$ and $w R v$, then there exists $v' \in W'$ such that $v E v'$ and $w' R' v'$
- If $w E w'$ and $w' R' v'$, then there exists $v \in W$ such that $v R v'$ and $w R v$

Intuition on Bisimulations over STSes



- More than just relating **STSes** in representation invariants per state.
- Bisimilar states can have different representation invariants.

Proof Indistinguishability

Knowing the proof of a client against the right (target STS conventionally π') *enables* deducing the proof against the bisimilar on the left (source STS conventionally π).

A Quick Tour on STS Assertions

- Invariants $\boxed{\varphi}^\gamma_\pi$, client capability $\boxed{s; T}^\gamma$

STSalloc

$$\varphi(s) \Rightarrow \exists \gamma. \boxed{\varphi}^\gamma_\pi * \boxed{s; \text{AllTokens} \setminus \mathcal{T}(s)}^\gamma$$

STSoPEN

$$\boxed{\varphi}^\gamma_\pi * \boxed{s; T}^\gamma \Rightarrow (\exists s'. \ulcorner (s_0, T) \rrcorner \sqsubseteq_\pi^{rely^*} (s', T)^\top * \varphi(s) * \forall s', T'. \ulcorner (s', T) \rrcorner \sqsubseteq_\pi^{guar.^*} (s', T')^\top * \varphi(s') \Rightarrow \boxed{s'; T'}^\gamma)$$

UPDIsl

α physically atomic

$$\frac{\forall s_0. ((s; T) \sqsubseteq_\pi^{rely^*} (s_0; T)) \vdash \{\varphi(s_0) * P\} \alpha \{\exists s', T'. (s_0; T) \sqsubseteq_\pi^{guar.^*} (s'; T') * \varphi(s') * Q\}}{\boxed{\varphi}^\gamma_\pi \vdash \{\boxed{s; T}^\gamma * P\} \alpha \{\exists s', T'. \boxed{s'; T'}^\gamma * Q\}}$$

Figure: Iris STS Library [Jung et al.(2015)] simplified with later modality and invariant masks omitted

Decomposing Bisimilarity in STSes

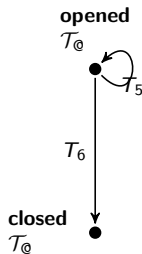
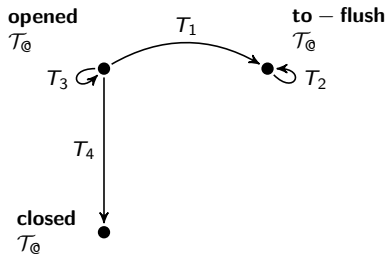
The bisimulation $(\mathcal{M}(\pi, \pi', \varphi, \varphi', s, T, U))$ between two state machines, π and π' is composed of

- The source STS – π
- The target STS – π'
- The source STS's state interpretation function – φ
- The target STS's state interpretation function – φ'
- Token Embedding – $\epsilon_S : \mathcal{S}(\pi) \mapsto \mathcal{S}(\pi')$
- State Embedding – $\epsilon_T : \mathcal{T}(\pi) \mapsto \mathcal{T}(\pi')$
- The Law of Rely
- The Law of Guarantee
- The Law of Tolerance
- The state of source STS from which bisimulation is considered against any client interference with the token set T

Proof Translation

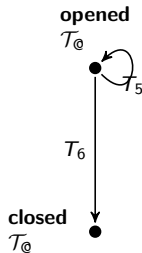
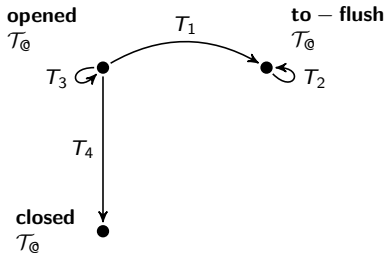
Obtain a proof rule utilizing the bisimulation to translate proofs between bisimilar state machines!

Embeddings for Tokens and States



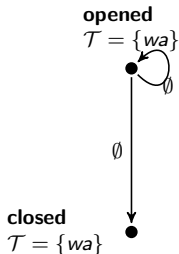
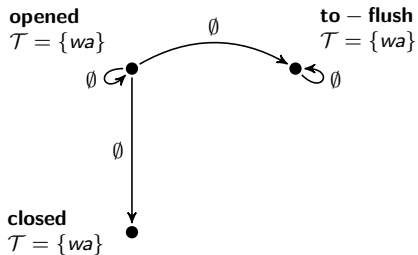
- How to impose *indistinguishability* of behavior observed on *bisimilar* states
- How to impose *indistinguishability* over the steps taken between *bisimilar* states

Embeddings for Tokens



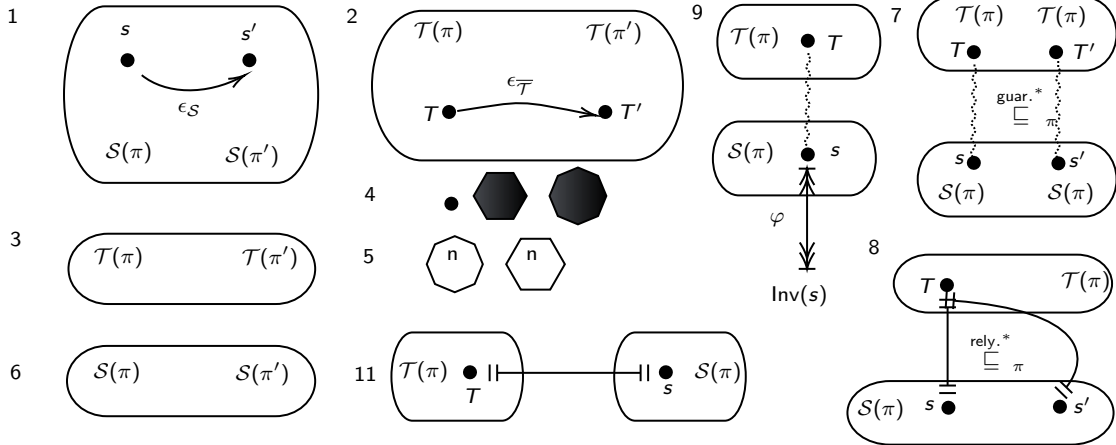
- Make *token exchange* align with *indistinguishability*: Embedding tokens (lifted $\epsilon_{\overline{T}}$) T_1 , T_3 , and T_2 to T_5 , and T_4 to T_6

Embeddings for State

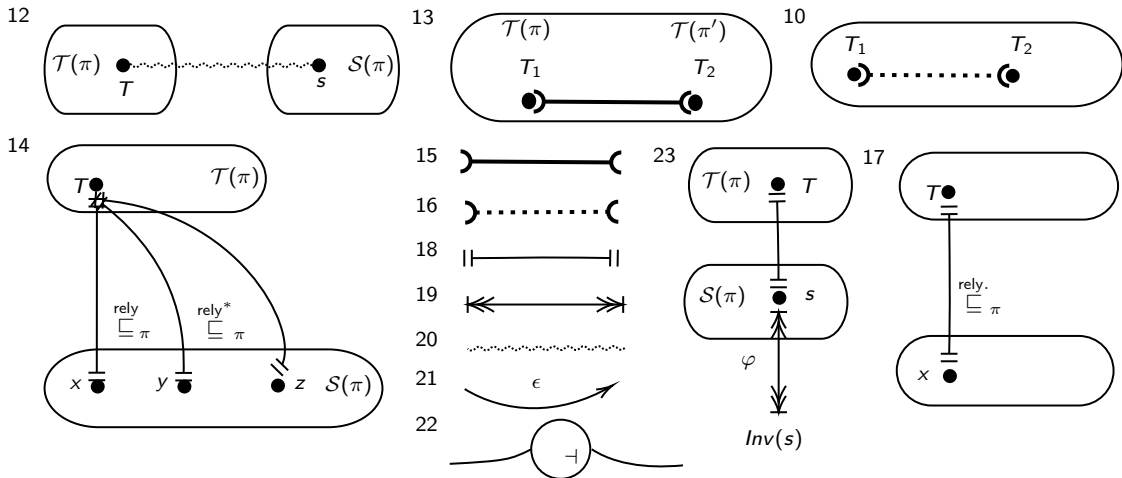


- Setting client interference aside, we would consider opened and flushed of the left state machine to be *bisimilar* to opened of the right state machine

A Quick Tour on the Representation of Laws - 1



A Quick Tour on the Representation of Laws - 2



We Need This

$$\begin{array}{c}
 \text{BISIM} \\
 \pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{[s; \overline{\epsilon_{\overline{T}}(\mathcal{T})}]_{\pi'}^\gamma * P\} \mathcal{C} \{[s'; \overline{T'}]_{\pi'}^\gamma * Q\} \\
 \hline
 [\varphi]_\pi^\gamma \vdash \{[q; \overline{T}]_\pi^\gamma * P\} \mathcal{C} \{[q'; \overline{T'}]_\pi^\gamma * Q\}
 \end{array}$$

We Use This

UPDISL

α physically atomic

$$\frac{\forall s_0 . ((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T)) \vdash \{\varphi(s_0) * P\} \alpha \{\exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \varphi(s') * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{s; T}^{\gamma} * P\} \alpha \{\exists s', T' . \boxed{s'; T'}^{\gamma} * Q\}}$$

BISIM

$$\frac{\pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{\boxed{s; \overline{T}(\mathcal{T})}^{\gamma}_{\pi'} * P\} C \{\boxed{s'; T'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{q; T}^{\gamma}_{\pi} * P\} C \{\boxed{q'; T'}^{\gamma}_{\pi} * Q\}}$$

How? Obligation - 1

UPDISL

α physically atomic

$$\forall s_0 . \quad \boxed{((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T))} \vdash \{\varphi(s_0) * P\} \alpha \{\exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \varphi(s') * Q\}$$

$$\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{s; T}_{\pi}^{\gamma} * P\} \alpha \{\exists s', T' . \boxed{s'; T'}_{\pi}^{\gamma} * Q\}$$

BISIM

$$\frac{\pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{\boxed{s; \overline{T}(T)}_{\pi'}^{\gamma} * P\} C \{\boxed{s'; T'}_{\pi'}^{\gamma} * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{q; T}_{\pi}^{\gamma} * P\} C \{\boxed{q'; T'}_{\pi}^{\gamma} * Q\}}$$

The Law of Rely

Theorem (The Law of Rely)

$$\begin{aligned} \forall s'. (s; T) \stackrel{\text{rely}^*}{\sqsubseteq}_{\pi} (s'; T) \leftrightarrow \\ (\forall_{s_1, s'_1, T_1}. \epsilon_S(s, s_1) \rightarrow \epsilon_S(s', s'_1) \rightarrow \epsilon_{\overline{T}}(T, T_1) \rightarrow (s_1; T_1) \stackrel{\text{rely}^*}{\sqsubseteq}_{\pi'} (s'_1; T_1)) \end{aligned}$$

- We do not drop any client interference with capabilities T
- Indetification of the states that are tolerant to the client interference from which the STS can take steps (Guarantee)
- Bookkeeping of the client interference needed!
- Identifying the valid *pre* state

The Law of Rely

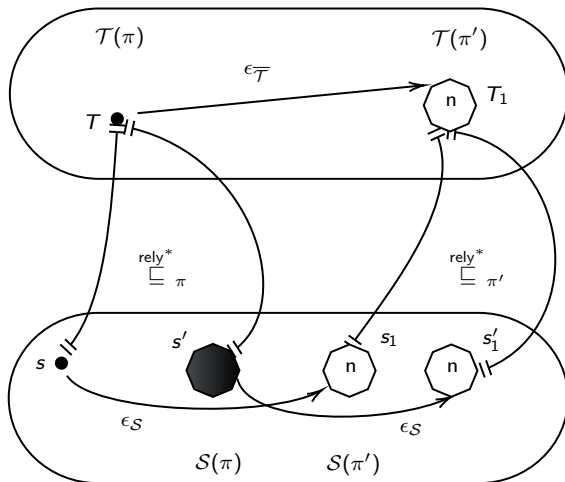
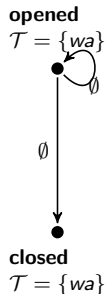
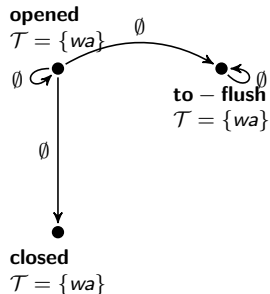
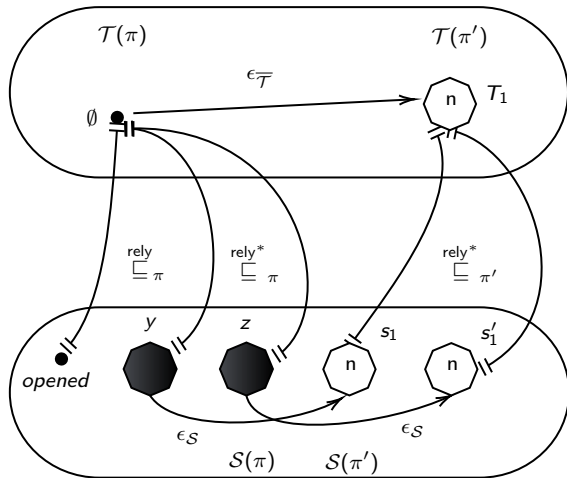


Figure: Embedding The Client Interface

The Law of Rely for a Bisimulation Instance



We Need This

UPDISL

α physically atomic

$$\frac{\forall s_0 . ((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T)) \vdash \{\varphi(s_0) * P\} \alpha \{\exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \varphi(s') * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{s; T}_{\pi}^{\gamma} * P\} \alpha \{\exists s', T' . \boxed{s'; T'}_{\pi}^{\gamma} * Q\}}$$

BISIM

$$\frac{\pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{\boxed{s; \overline{T}(\mathcal{T})}_{\pi'}^{\gamma} * P\} C \{\boxed{s'; T'}_{\pi'}^{\gamma} * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{q; T}_{\pi}^{\gamma} * P\} C \{\boxed{q'; T'}_{\pi}^{\gamma} * Q\}}$$

How? Obligation - 2

UPDISL

α physically atomic

$$\forall s_0 . ((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T)) \vdash \{\varphi(s_0) * P\} \alpha \{\exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \varphi(s') * Q\}$$

$$\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{s; T}_{\pi}^{\gamma} * P\} \alpha \{\exists s', T' . \boxed{s'; T'}_{\pi}^{\gamma} * Q\}$$

BISIM

$$\frac{\pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{\boxed{s; \overline{T}(T)}_{\pi'}^{\gamma} * P\} C \{\boxed{s'; T'}_{\pi'}^{\gamma} * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{q; T}_{\pi}^{\gamma} * P\} C \{\boxed{q'; T'}_{\pi}^{\gamma} * Q\}}$$

The Law of Guarantee

Theorem (Guarantee Bisim without Invariants)

$$\begin{aligned} \forall_{q', q, T'}. \epsilon_{\overline{T}}(T) \equiv T' \rightarrow \epsilon_S(s, q) \rightarrow (q; T') &\stackrel{\text{rely}^*}{\sqsubseteq} \pi' (q'; T') \rightarrow \\ \forall_{q'', T''}. (q'; T') &\stackrel{\text{guar.}}{\sqsubseteq} \pi' (q''; T'') \rightarrow \\ \exists_{s', s'', T'_0, T''_0}. (s'; T'_0) &\stackrel{\text{guar.}}{\sqsubseteq} \pi (s''; T''_0) \wedge \\ \epsilon_S(s') = q' \wedge \epsilon_S(s'') = q'' \wedge \epsilon_{\overline{T}}(T'_0) \equiv T' \wedge \epsilon_{\overline{T}}(T''_0) \equiv T'' \end{aligned}$$

- Under the embedded client interference, the steps taken by the target STS must be countered by a one in the source STS
- From target STS to source STS
- Identifying the valid *post* state

The Law of Guarantee

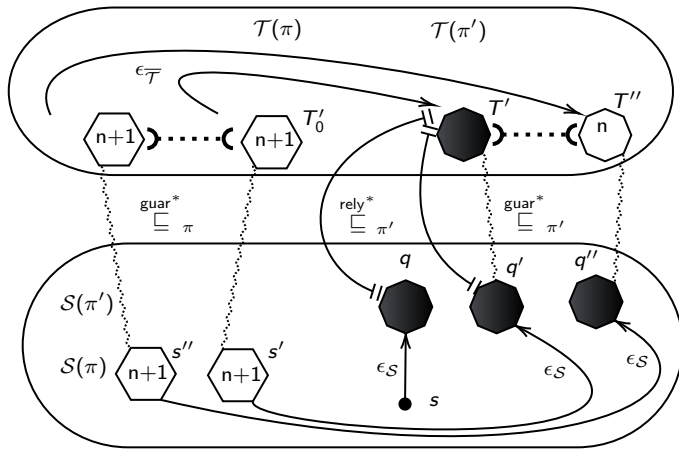
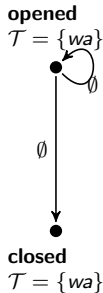
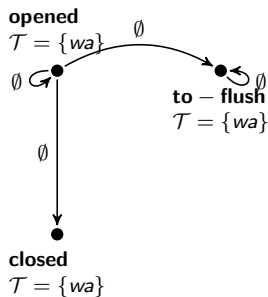
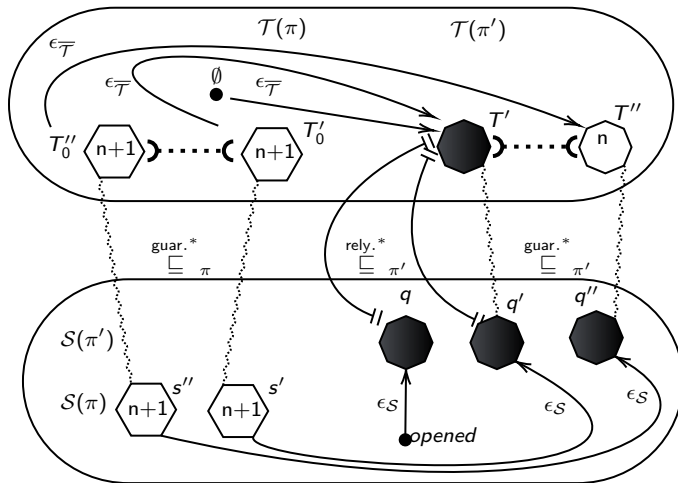
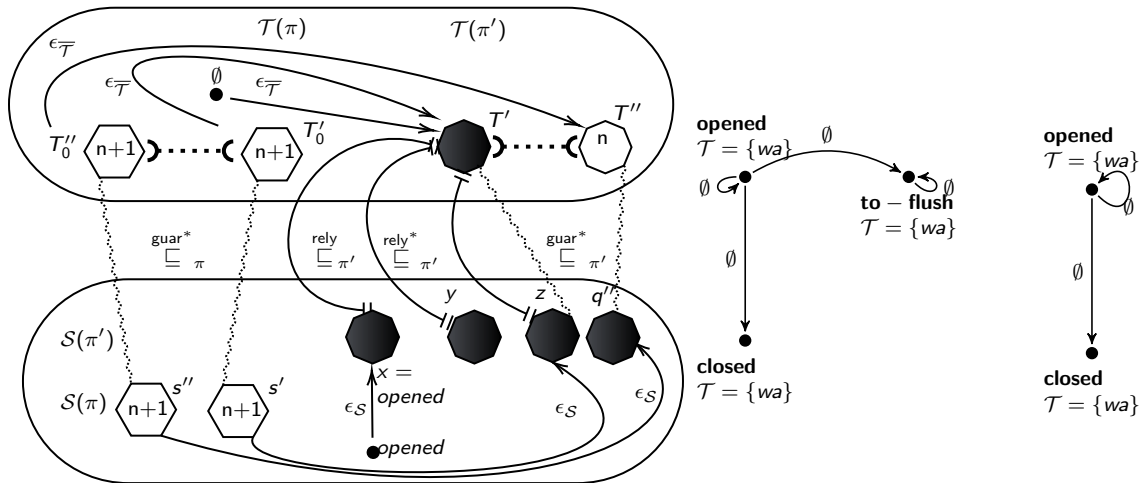


Figure: Embedding the Guarantee Steps of Target STS

The Law of Guarantee for the Bisimulation Instance



The Law of Guarantee for the Bisimulation Instance



We Need This

UPDISL

α physically atomic

$$\frac{\forall s_0 . ((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T)) \vdash \{\varphi(s_0) * P\} \alpha \{\exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \varphi(s') * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{s; T}_{\pi}^{\gamma} * P\} \alpha \{\exists s', T' . \boxed{s'; T'}_{\pi}^{\gamma} * Q\}}$$

BISIM

$$\frac{\pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{\boxed{s; \overline{T}}_{\pi'}^{\gamma} * P\} C \{\boxed{s'; T'}_{\pi'}^{\gamma} * Q\}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{\boxed{q; T}_{\pi}^{\gamma} * P\} C \{\boxed{q'; T'}_{\pi}^{\gamma} * Q\}}$$

How? Obligation - 3

UPDISL

$$\forall s_0 . ((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T)) \vdash \{ \boxed{\varphi(s_0)} * P \} \alpha \{ \exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \varphi(s') * Q \}$$

α physically atomic

$$\boxed{\varphi}_{\pi}^{\gamma} \vdash \{ \boxed{s; T}_{\pi}^{\gamma} * P \} \alpha \{ \exists s', T' . \boxed{s'; T'}_{\pi}^{\gamma} * Q \}$$

BISIM

$$\frac{\pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{ \boxed{s; \overline{T}(T)}_{\pi'}^{\gamma} * P \} C \{ \boxed{s'; T'}_{\pi'}^{\gamma} * Q \}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{ \boxed{q; T}_{\pi}^{\gamma} * P \} C \{ \boxed{q'; T'}_{\pi}^{\gamma} * Q \}}$$

How? Obligation - 4

UPDISL

$$\begin{array}{c}
 \alpha \text{ physically atomic} \\
 \forall s_0 . ((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T)) \vdash \{ \boxed{\varphi(s_0)} * P \} \alpha \{ \exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \boxed{\varphi(s')} * Q \} \\
 \hline
 \boxed{\varphi}_{\pi}^{\gamma} \vdash \{ \boxed{s; T}_{\pi}^{\gamma} * P \} \alpha \{ \exists s', T' . \boxed{s'; T'}_{\pi}^{\gamma} * Q \}
 \end{array}$$

BISIM

$$\begin{array}{c}
 \pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{ \boxed{s; \overline{T}}_{\pi}^{\gamma} * P \} C \{ \boxed{s'; \overline{T}'}_{\pi'}^{\gamma} * Q \} \\
 \hline
 \boxed{\varphi}_{\pi}^{\gamma} \vdash \{ \boxed{q; \overline{T}}_{\pi}^{\gamma} * P \} C \{ \boxed{q'; \overline{T}'}_{\pi}^{\gamma} * Q \}
 \end{array}$$

How? Intuition on Invariants in Bisimulation

UPDISL

$$\begin{array}{c} \alpha \text{ physically atomic} \\ \forall s_0 . ((s; T) \sqsubseteq_{\pi}^{\text{rely}^*} (s_0; T)) \vdash \{ \boxed{\varphi(s_0)} * P \} \alpha \{ \exists s', T' . (s_0; T) \sqsubseteq_{\pi}^{\text{guar}^*} (s'; T') * \boxed{\varphi(s')} * Q \} \\ \hline \boxed{\varphi}_{\pi}^{\gamma} \vdash \{ \boxed{s; T}_{\pi}^{\gamma} * P \} \alpha \{ \exists s', T' . \boxed{s'; T'}_{\pi}^{\gamma} * Q \} \end{array}$$

BISIM

$$\frac{\pi \sim \pi' \quad q \in_S s \quad q' \in_S s' \quad \{ \boxed{s; \overline{\epsilon_T(T)}}_{\pi'}^{\gamma} * P \} C \{ \boxed{s'; \overline{\epsilon_{T'}(T')}}_{\pi'}^{\gamma} * Q \}}{\boxed{\varphi}_{\pi}^{\gamma} \vdash \{ \boxed{q; \overline{T}}_{\pi}^{\gamma} * P \} C \{ \boxed{q'; \overline{T'}}_{\pi}^{\gamma} * Q \}}$$

- The post condition of the target state machine π' implies the post condition of the source state machine π , e.g., opened
- Applying UPDISL would give
 - $\varphi'(\epsilon_S(s_0)) \vdash \exists s' . \varphi'(s')$
 - and precondition in the source STS – $\varphi(s_0)$
- To obtain $\exists s' . \varphi(s')$

Invariants of File Protocols

Definition (File Protocol Invariants)

$$\varphi_{\text{distributedfile}}(\ell, R)(s) \triangleq \left\{ \begin{array}{ll} \text{match } s \text{ with} \\ \text{to } - \text{ flush} \Rightarrow & R * \exists \text{fs. isValidDirty(fs)} * \\ & \ell \mapsto (\text{fs.id}, \text{fs.status} = \text{dirty}) \\ \text{opened} \Rightarrow & R * \exists \text{fs. isValid(fs)} * \\ & \ell \mapsto (\text{fs.id}, \text{fs.status} = \text{clean}) \\ \text{closed} \Rightarrow & \exists \text{fs. isValidClosed(fs)} * \\ & \ell \mapsto (\text{fs.id}, \text{fs.status} = \text{closed}) \end{array} \right\}$$

$$\varphi_{\text{file}}(\ell, R)(s) \triangleq \left\{ \begin{array}{ll} \text{match } s \text{ with} \\ \text{opened} \Rightarrow & R * \exists \text{fs. isValid(fs)} * \ell \mapsto (\text{fs.id}, \text{fs.status} = \text{clean} \vee \text{dirty}) \\ \text{closed} \Rightarrow & \exists \text{fs. isValidClosed(fs)} * \ell \mapsto (\text{fs.id}, \text{fs.status} = \text{closed}) \end{array} \right\}$$

The Law of Tolerance

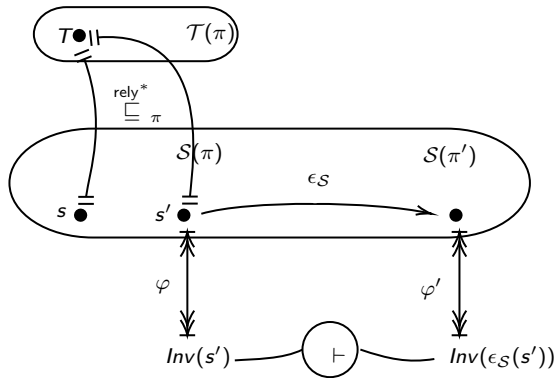


Figure: The Law of Tolerance for a Valid Pre-Condition

Theorem (The Law of Tolerance)

$$\forall s' . (s \text{ T}) \stackrel{\text{rely}^*}{\sqsubseteq} \pi (s'; \text{T}) \leftrightarrow \boxed{\varphi(s') \vdash \varphi'(\epsilon_S(s'))}$$

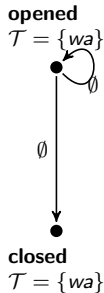
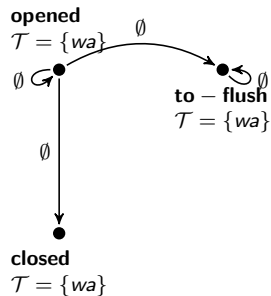
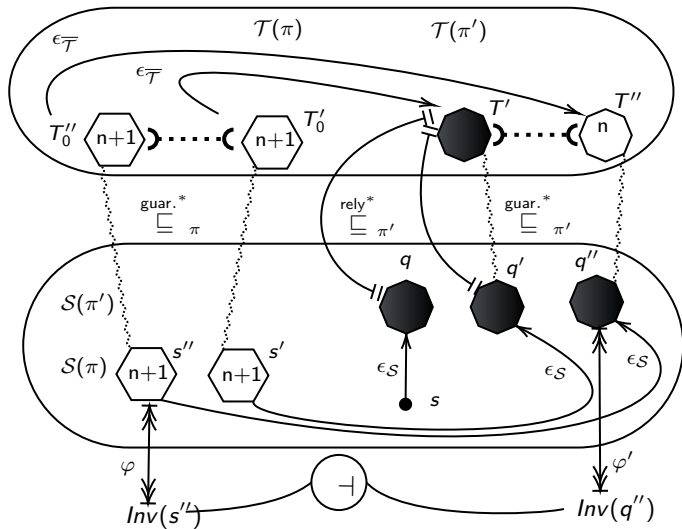
- The source STS's precondition for the embedded states is stronger
- $\varphi(\text{flushed})$ of π or $\varphi(\text{opened})$ of π implies $\varphi'(\text{opened})$ of π'

Revisiting the Law of Guarantee for Invariants

Theorem (The Law of Guarantee with Invariants)

$$\begin{aligned} \forall q' . (\epsilon_S(s); \epsilon_{\overline{T}}(T)) &\stackrel{\text{rely}^*}{\sqsubseteq} \pi' (\epsilon_S(q'); \epsilon_{\overline{T}}(T)) \rightarrow \\ \forall q'', T'' . (q'; \epsilon_{\overline{T}}(T)) &\stackrel{\text{guar}^*}{\sqsubseteq} \pi' (q''; T'') \rightarrow \\ \exists s' s'' T0' T0'' . (s'; T0') &\stackrel{\text{guar}^*}{\sqsubseteq} \pi (s''; T0'') \wedge \\ &\epsilon_S(s') = q' \wedge \epsilon_S(s'') = q'' \wedge \\ &(\epsilon_{\overline{T}}(T0')) = (\epsilon_{\overline{T}}(T)) \wedge (\epsilon_T(T0')) = T'' \wedge \\ &\boxed{\varphi'(q'') \vdash \varphi(s'')} \end{aligned}$$

Revisiting the Law of Guarantee for Invariants

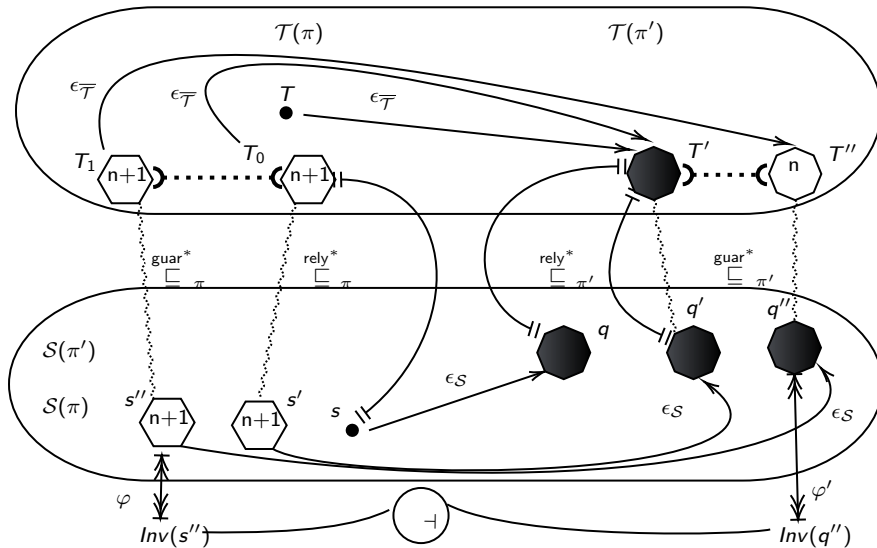


Soundness

Theorem (Soundness)

The updated state from UPDISL is preserved by the bisimulation.

Soundness



Keeping Promises

TRANSFER FILE WRITE

$$\frac{\begin{array}{c} \pi \sim \pi' \quad \text{opened} \in_S s \quad q' \in_S s' \\ \{[s; \epsilon_{\overline{T}}(\mathcal{T})]_{\pi'}^\gamma * P\} \text{ write } \ell \text{ new_val } \{[s'; \overline{T'}]_{\pi'}^\gamma * Q\} \end{array}}{\boxed{\varphi_{\text{distributedfile}}}_{\pi}^\gamma \vdash \{[\text{opened}; \overline{T}]_{\pi}^\gamma * P\} C \{[q'; \overline{T'}]_{\pi}^\gamma * Q\}}$$

Ongoing Work

- Bisimulation over Restricted Submodels
 - We need to strengthen the bisimulation
 - What is this strength?
 - Guarantee steps taken at the target machine with $T \setminus U$ has to be indistinguishable (entail) the ones taken with T
 - **The Law of Abduction**: Knowing more about U and capabilities dropped with the absence of it.
- Extending Rules for the Complete Iris `HEAPLANG`
- An abstract framework with relational *embeddings* is finished and written.
 - Subjectively more promising!
 - We need to justify concretely via an example, that is, an example exposes incapability of current non-relational embeddings of states and tokens

The Future Directions

- Exploit obvious application fields, e.g., device drivers
- Only Iris pluggable?



Tej Chajed. 2022.

Verifying a concurrent, crash-safe file system with sequential reasoning.

Ph.D. Dissertation. Machetutes Institute of Technology, Cambridge, MA.

Available at <https://dspace.mit.edu/handle/1721.1/144578>.



Tej Chajed, Joseph Tassarotti, M. Frans Kaashoek, and Nickolai Zeldovich. 2019.

Verifying concurrent, crash-safe systems with Perennial. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) (*SOSP '19*). Association for Computing Machinery, New York, NY, USA, 243–258.

<https://doi.org/10.1145/3341301.3359632>



Hoang-Hai Dang, Jacques-Henri Jourdan, Jan-Oliver Kaiser, and Derek Dreyer. 2019.

RustBelt meets relaxed memory.

Proc. ACM Program. Lang. 4, POPL, Article 34 (Dec. 2019), 29 pages.

<https://doi.org/10.1145/3371102>



Marko Doko and Viktor Vafeiadis. 2016.

A Program Logic for C11 Memory Fences. In *Proceedings of the 17th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 9583* (St. Petersburg, FL, USA) (*VMCAI 2016*). Springer-Verlag, Berlin, Heidelberg, 413–430.

Springer-Verlag, Berlin, Heidelberg, 413–430.

https://doi.org/10.1007/978-3-662-49122-5_20



Marko Doko and Viktor Vafeiadis. 2017.

Tackling Real-Life Relaxed Concurrency with FSL++. In *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on*

Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings (Uppsala, Sweden). Springer-Verlag, Berlin, Heidelberg, 448–475.

https://doi.org/10.1007/978-3-662-54434-1_17



George Edward Hughes and Max J Cresswell. 1996.

A new introduction to modal logic.



Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015.

Iris: Monoids and Invariants As an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Mumbai, India) (POPL '15). ACM, New York, NY, USA, 637–650.

<https://doi.org/10.1145/2676726.2676980>



Ismail Kuru and Colin S. Gordon. 2024.

Modal Abstractions for Virtualizing Memory Addresses.

arXiv:2307.14471 [cs.PL] <https://arxiv.org/abs/2307.14471>



Richard McDougall and Jim Mauro. 2006.

Solaris internals: Solaris 10 and OpenSolaris kernel architecture.

Pearson Education.



Josep Tassarotti Tej Chajed and contributors. 2023.

Post-crash modality in Perennial's Coq Mechanization.

https://github.com/mit-pdos/perennial/blob/master/src/goose_lang/crash_modality.v



Aaron Turon, Derek Dreyer, and Lars Birkedal. 2013.

Unifying Refinement and Hoare-Style Reasoning in a Logic for Higher-Order Concurrency. In *ICFP*.



Simon Friis Vindum, Aïna Linn Georges, and Lars Birkedal. 2025.

The Nextgen Modality: A Modality for Non-Frame-Preserving Updates in Separation Logic. In *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Denver, CO, USA) (CPP '25). Association for Computing Machinery, New York, NY, USA, 83–97.

<https://doi.org/10.1145/3703595.3705876>



Andrew Wagner, Zachary Eisbach, and Amal Ahmed. 2024.

Realistic Realizability: Specifying ABIs You Can Count On.

Proc. ACM Program. Lang. 8, OOPSLA2, Article 315 (Oct. 2024), 30 pages.

<https://doi.org/10.1145/3689755>

The End