Modal Abstractions for OS Kernels

Ph.D. Defense

Ismail Kuru
Department of Computer Science
Drexel University

March 14, 2025

Overview

Overview

Part I

Modal Understanding of Location Virtualization

The Essentials in Systems Programming

$$\underbrace{\text{pointer va}}_{\text{a virtual reference}} := \overbrace{\text{malloc (size)}}^{\text{a supposedly allocated physical resource}}$$

Memory Location Virtualization

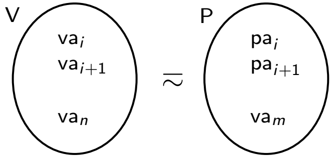


Figure: Virtualization: The Deception of Abundance

Memory Location Virtualization: Abstraction

An Address Space with Logical Name γ

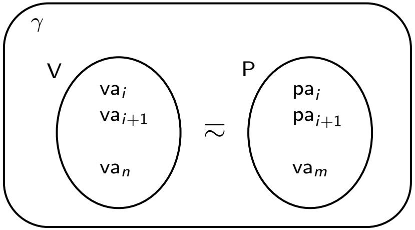


Figure: Address-Spaces: Named Containers for Virtual Memory Mappings

A Program Named γ_n

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

A Program Named 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

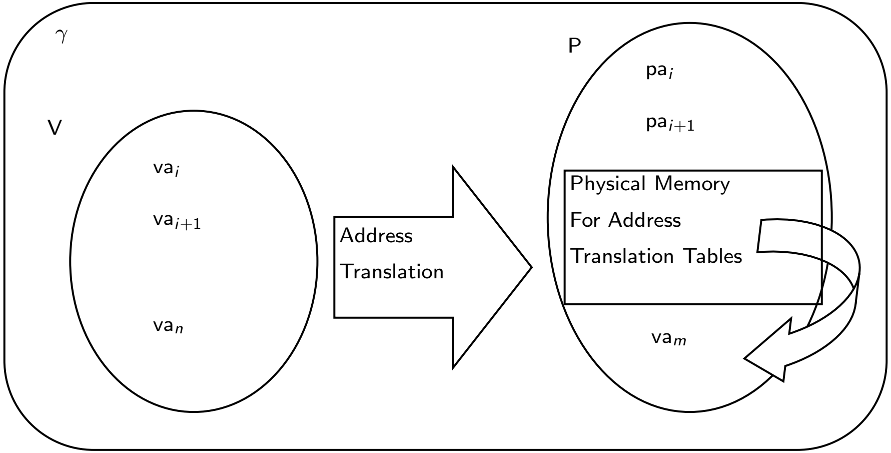


Figure: Address-Translation: A Mechanism for Realizing Address Virtualization

Page Tables

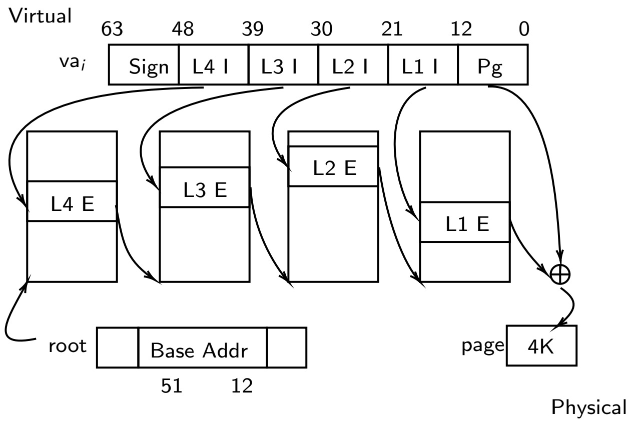


Figure: Page-Tables (**PT**): Data Structures for Address-Translation

A Complete Picture of Address-Space Abstraction

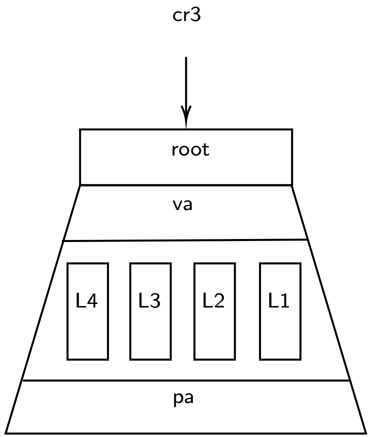


Figure: Depicting an Address-Space with its Essential Aspects

The Current View of Memory

The register cr_3 points to the current view of the memory, i.e., the loaded address space in the memory

.

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

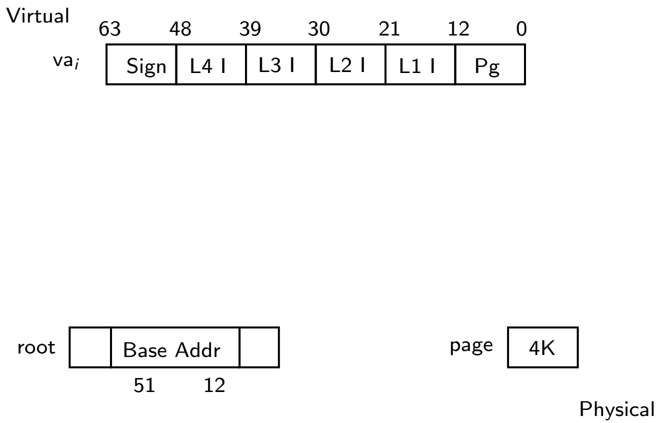


Figure: Physical and Virtual Resources

Sharing Physical Page Tables

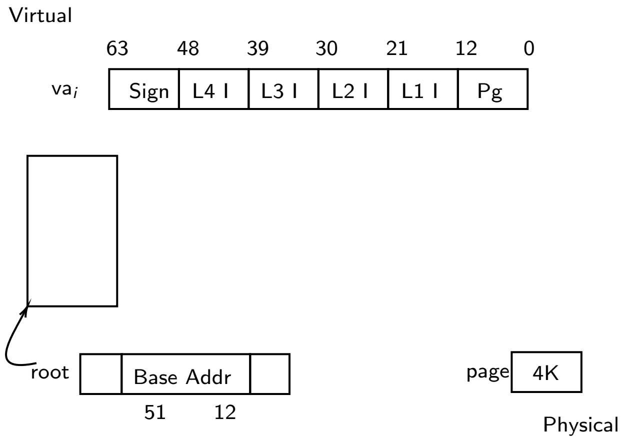


Figure: Traversal Starts from the root

```
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){
```

Sharing Physical Page Tables

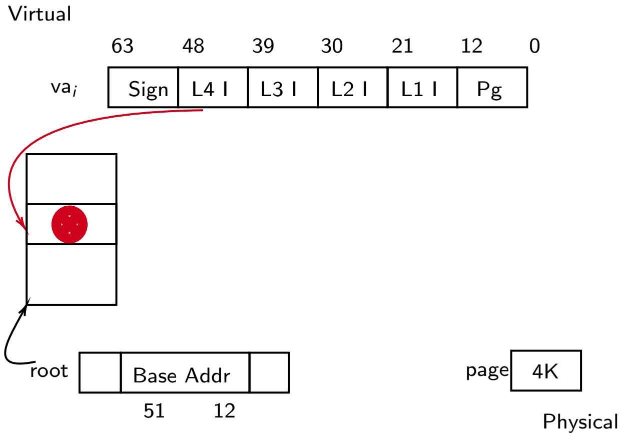


Figure: Missing L4 Entry

```
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)];
```

Sharing Physical Page Tables

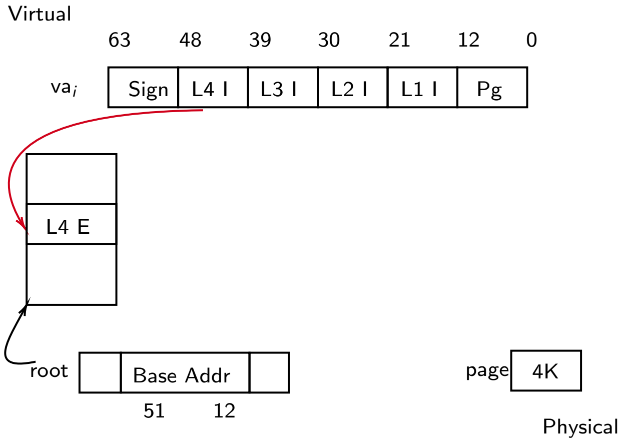


Figure: A New L4 Entry

```
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)];
```

Sharing Physical Page Tables

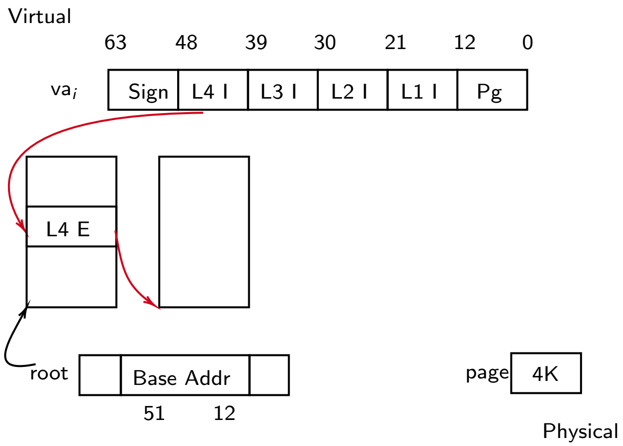


Figure: Accessing to Table L3

```
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);
}
```

Sharing Physical Page Tables

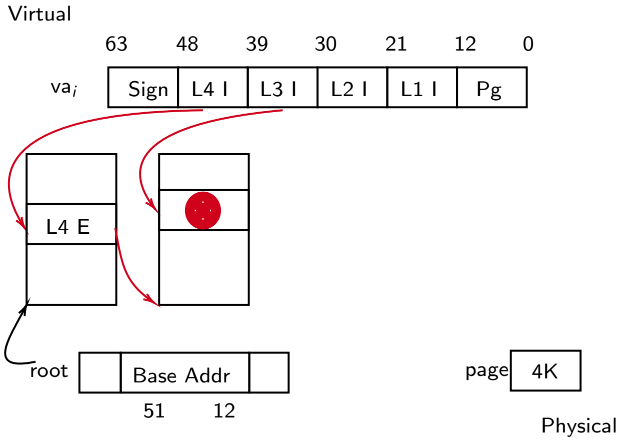


Figure: Missing L3 Entry

```
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);
}
```

Sharing Physical Page Tables

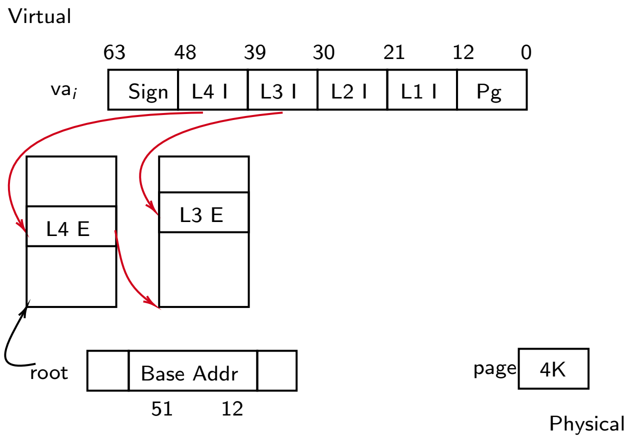
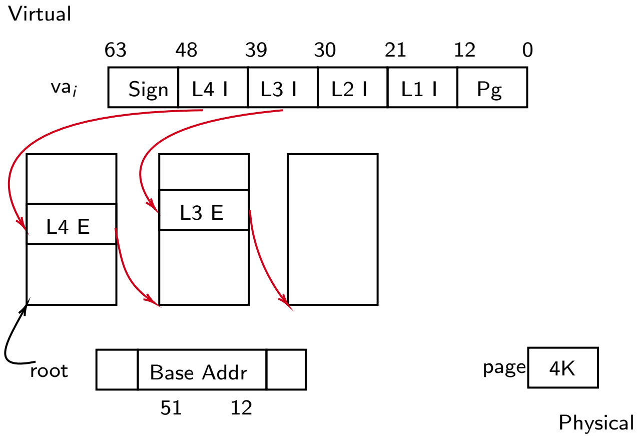


Figure: A New L3 Entry

```
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)];
}
```

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

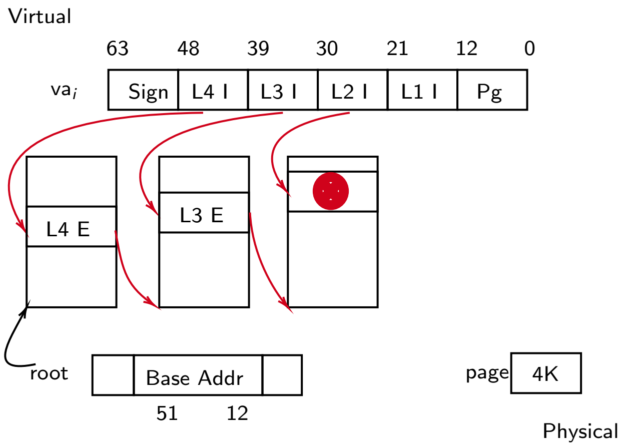


Figure: Missing L2 Entry

```
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);
}
```

Sharing Physical Page Tables

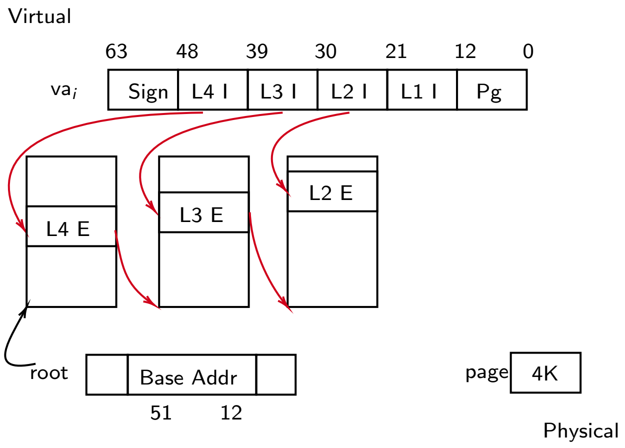


Figure: A New L2 Entry

```
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 = &l3[L2I(va)];
}
```

Sharing Physical Page Tables

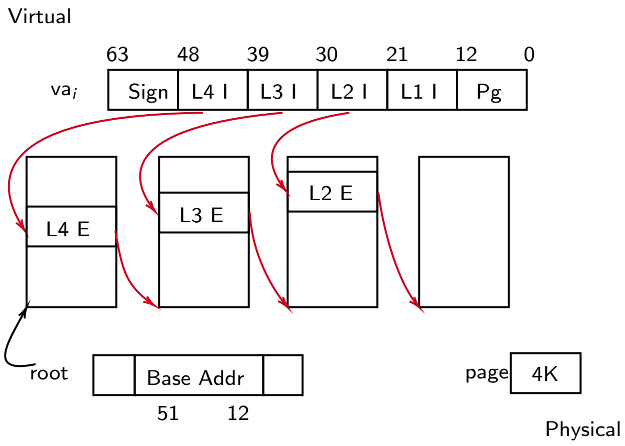


Figure: Accessing L1 Table

```
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 = &l3[L2I(va)];
 pte_t *l1 = pte_nxt_table(l2_entry);
}
```

Sharing Physical Page Tables

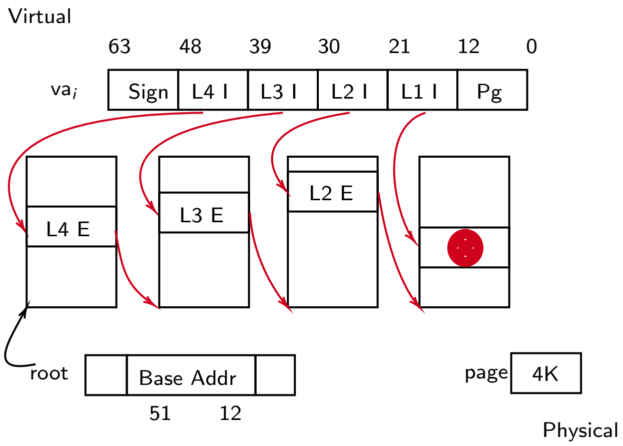


Figure: Missing L1 Entry

```
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 = &l3[L2I(va)];
 pte_t *l1 = pte_nxt_table(l2_entry);
}
```

Sharing Physical Page Tables

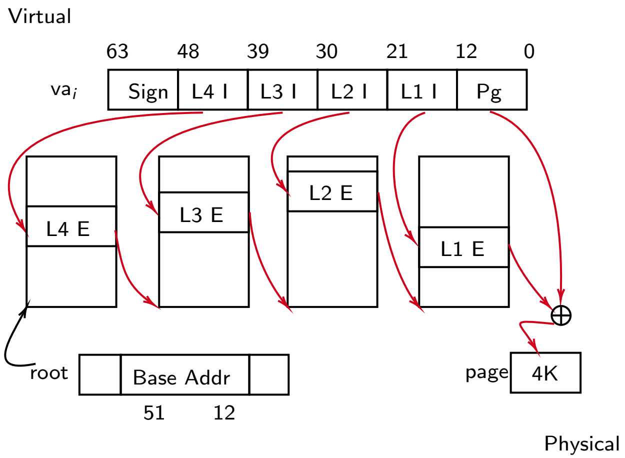


Figure: Accessing to the Page Referenced by L1 Entry

```
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);
 pte_t *l1_entry = &l1[L1I(va)];
 return l1_entry
}
```

Breaking Soundness in Sharing

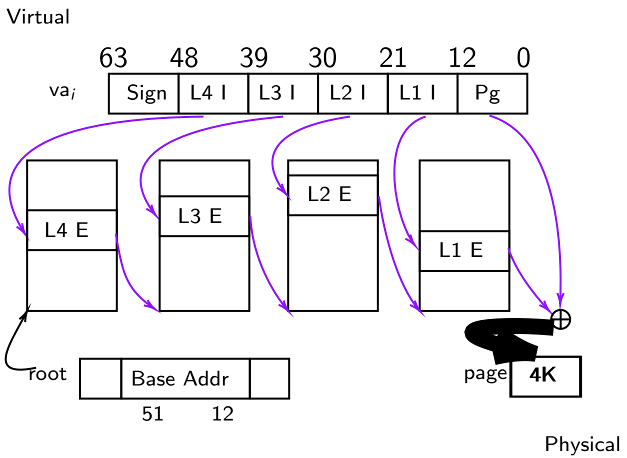
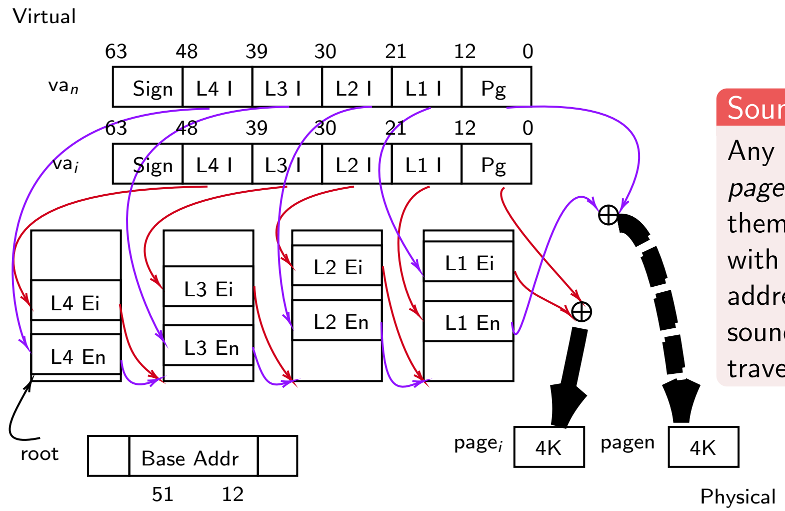


Figure: Sharing Page-Tables under Updates to Page-Tables

Breaking Soundness in Sharing



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

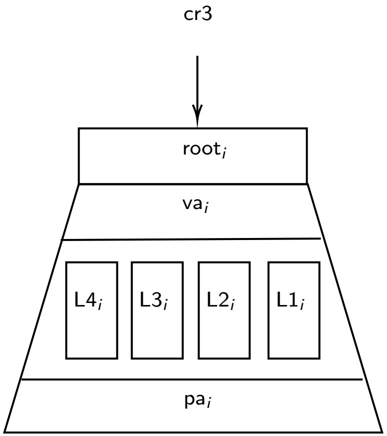


Figure: An Address Space with Unique Root Address $root_i$

Managing Agnostic Memory Mappings

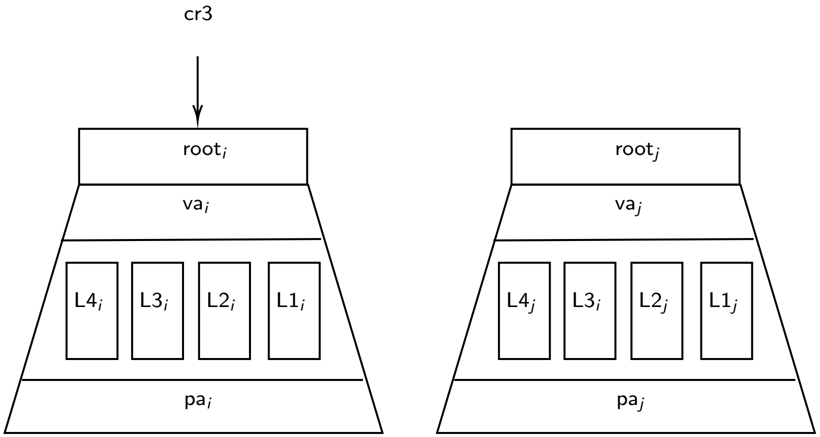


Figure: Two Address-Spaces with the Unique Root Addresses root_i and root_j

Managing Agnostic Memory Mappings

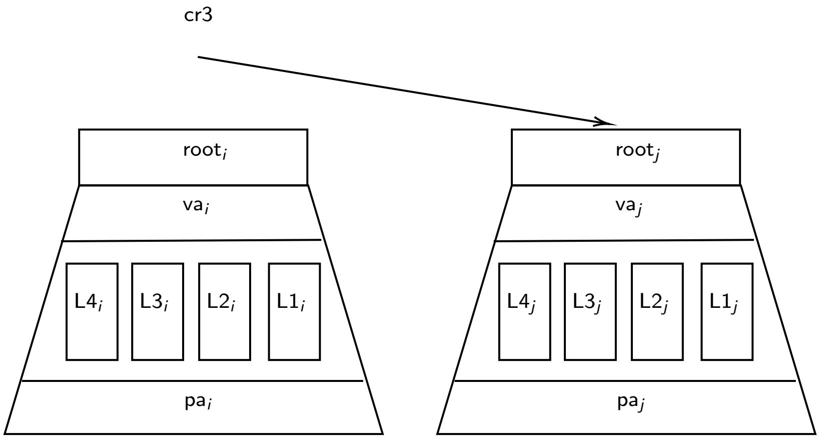


Figure: Switching Address-Spaces

Managing Agnostic Memory Mappings

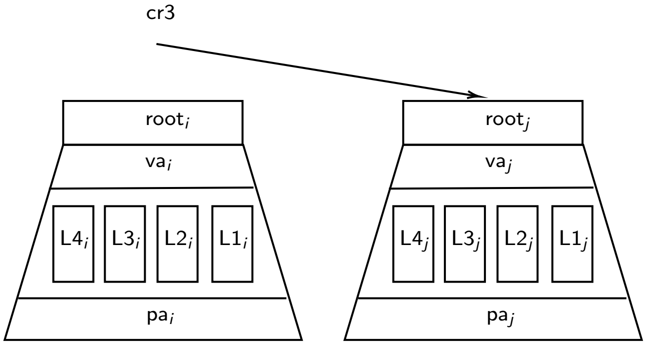


Figure: Switching Address-Spaces

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 Conjuction

$$\frac{\{P\}\ e\ \{Q\}}{\{P * R\}\ e\ \{Q * R\}}\ \textsc{Frame}$$

Separation Logic: Ownership

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

Separation Logic: Invariants

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

Defining Some Ownersip 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?

$$\mathrm{L_4\_L_1\_PointsTo(va,\ l4e,\ l3e,\ l2e,\ l1e,\ paddr)} + \mathit{paddr} \mapsto_p \mathrm{page\_val}$$

Tokens for Traversals

$$\underbrace{\mathsf{va} \hookrightarrow^{\delta}_{\mathsf{q}} \mathsf{pa}}_{\text{Ghost translation}} * \underbrace{\mathsf{pa} \mapsto_{\mathsf{p}} \{\mathsf{qfrac}\}\ \mathsf{val}}_{\text{Physical location}}$$

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

A Candidate for Kernel Invariant

$$\mathcal{I}\mathsf{ASpace}(\theta, m) \triangleq \mathsf{ASpace\_Lookup}(\theta, m) * \mathop{\ast}_{(\mathsf{va},\mathsf{paddr}) \in \theta} \exists\, (\mathsf{l4e}\ \mathsf{l3e}\ \mathsf{l2e},\ \mathsf{l1e},\ \mathsf{paddr}).\, \mathsf{L_4\_L_1\_PointsTo}(\mathsf{va},\ \mathsf{l4e},\ \mathsf{l3e},\ \mathsf{l2e},\ \mathsf{l1e},\ \mathsf{paddr})$$

where

$$\mathsf{ASpace\_Lookup}(\theta, m) \triangleq \lambda\ \mathsf{cr3val}.\ \exists \delta\ .\ \ulcorner m\ !!\ \mathsf{cr3val} = \mathsf{Some}\ \delta \urcorner * \mathcal{A}\mathsf{bsPTableWalk}(\delta, \theta)$$

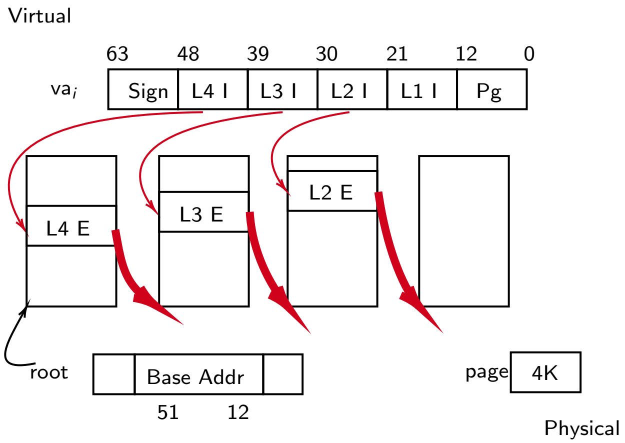
Figure: Global Address-Space Invariant with a fixed global map of address-space names m

A Candidate for Kernel Invariant

The Kernel Invariant in Action

How useful (complete) is this invariant?

Kernels Locating L1 Entries



```
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)

$$\mathcal{I}\mathsf{ASpace}_{\mathsf{id}}(\theta, \Xi, m) \triangleq \mathsf{ASpace\_Lookup}_{\mathsf{id}}(\theta, \Xi, m) * \mathsf{GhostMap}(\mathsf{id}, \Xi) *$$

$$\left(\mathop{\Huge *}_{(\mathsf{va},\mathsf{paddr}) \in \theta} \exists\, (\mathsf{l4e},\ \mathsf{l3e},\ \mathsf{l2e},\ \mathsf{l1e},\ \mathsf{paddr}).\ \mathsf{L_4\_L_1\_PointsTo}(\mathsf{va},\ \mathsf{l4e},\ \mathsf{l3e},\ \mathsf{l2e},\ \mathsf{l1e},\ \mathsf{paddr}) \right) *$$

$$\mathop{\Huge *}_{(\mathsf{pa},\mathsf{level}) \in \Xi} \exists\, (\mathsf{qfrac},\ \mathsf{q},\ \mathsf{val},\mathsf{va}).\ \ulcorner \mathsf{va} = \mathsf{pa} + \mathsf{KERNBASE}\ \mathsf{level} > 1 \urcorner * \underbrace{\mathsf{va} \hookrightarrow^{\delta}_{\mathsf{q}} \mathsf{pa}}_{\text{Ghost translation}} * \underbrace{\mathsf{pa} \mapsto_{\mathsf{p}} \{\mathsf{qfrac}\}\ \mathsf{val}}_{\text{Physical location}} *$$

$$\underbrace{\ulcorner \mathsf{qfrac} = 1 \leftrightarrow \neg \mathsf{entry\_present}\ (\mathsf{val}) \urcorner}_{\text{Entry validity}} *$$

$$\underbrace{\left(\ulcorner \mathsf{present\_L}(\mathsf{val}, \mathsf{level}) \urcorner \mathrel{-\!\!*} \forall_{\mathsf{i} \in 0..511}.\ ((\mathsf{entry\_page}\ \mathsf{val}) + \mathsf{i} * 8) \hookrightarrow^{\mathsf{id}} \mathsf{level\text{-}1} \right)}_{\text{Indexing into next level of tables}}$$

where

$$\mathsf{present\_L}(\mathsf{val}, \mathsf{level}) \triangleq \mathsf{entry\_present}(\mathsf{val}) \land \mathsf{level} > 0$$

Specifying P2V

$\{P * \mathcal{I}\texttt{ASpace}_{\text{id}}(\theta, \Xi \setminus \{\text{entry}\}), m) * \text{rbp-8} \mapsto_{\text{v}} \text{entry} * \texttt{rcx} \mapsto_{\text{r}} \_ * \text{entry} \mapsto_{\text{id}} \_ * \text{rtv} \hookrightarrow^{\delta s} \delta\}_{\text{rtv}}$

$\{\text{entry+KERNBASE} \mapsto_{\text{vpte,qfrac}} (\text{pte\_initialized (entry\_val.pfn)})^{\urcorner}\}_{\text{rtv}}$

$\{\texttt{rbp-16} \mapsto_{\text{v}} (\text{pte\_initialized (entry\_val.pfn)})) * \texttt{rax} \mapsto_{\text{r}} \text{table\_root (pte\_initialize(entry\_val.pfn))}\}_{\text{rtv}}$

$\{\forall_{i\in 0 \ldots 511}. ((\text{table\_root (pte\_initialized (entry\_val.pfn)))}) + \text{i * 8}) \hookrightarrow^{\text{id}} \text{v-1}\}$

```
;; uintptr_t next_virt_addr = (uintptr_t) P2V(entry.pfn<<12);
movabs KERNBASE, rcx
```
$\{\ldots * \texttt{rcx} \mapsto_{\text{r}} \text{KERNBASE} * \ldots\}_{\text{rtv}}$
```
add     rcx, rax
```
$\{\ldots * \texttt{rax} \mapsto_{\text{r}} \text{table\_root (pte\_initialize(entry\_val.pfn))} + \text{KERNBASE} * \ldots\}_{\text{rtv}}$
```
...  ;;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

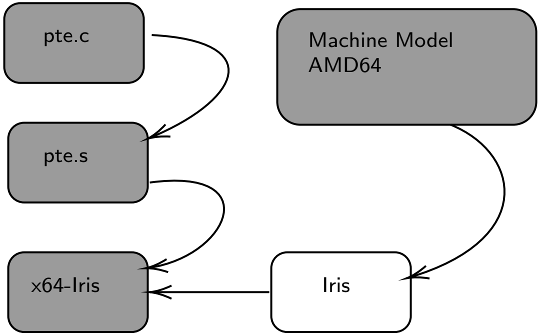


Figure: x64-Iris

- 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)]	$\Diamond\ P$	$\ell \mapsto_{n}^{\overline{\gamma}} v$	Strong	Crash Recovery
NextGen Modality [Vindum et al.(2025)]	$\overset{t}{\hookrightarrow}\ P$	Own (t(a))	Strong	Determined Based on the Model*
StackRegion Modality* [Vindum et al.(2025)]	$\overset{ICut^{n}}{\hookrightarrow}\ P$	$\boxed{\text{n}}\ \ell \mapsto v$	Strong	Alloc and Return to/from stack
Memory-Fence Modality [Doko and Vafeiadis(2016), Doko and Vafeiadis(2017), Dang et al.(2019)]	$\triangle_{\pi}$ and $\triangledown_{\pi}$	$\ell \mapsto v$	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

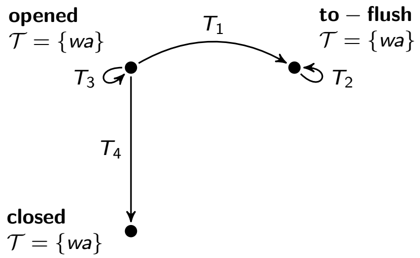


Figure: STS for Distributed File Protocol

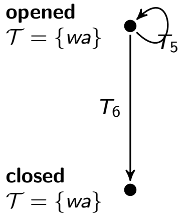


Figure: STS for Traditional File Protocol

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 $\mathcal{T} : \mathcal{S} \to \mathsf{TokSet}$,
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 $\varphi : \mathcal{S} \to \mathsf{Prop}$.

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
- $V : \text{PropVar} \to \mathcal{P}(W)$ gives for each propositional variable p a set of worlds $V(p)$ where p is considered true

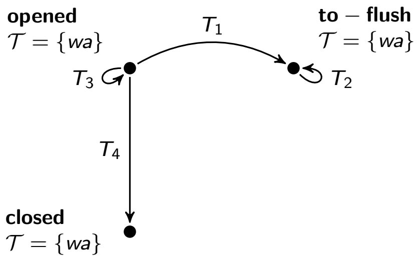
Bisimulations over Kripke Models

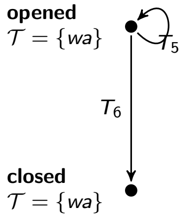
Definition ((Propositional) Bisimulation of Kripke Structures: $\mathfrak{M} \sim \mathfrak{M}'$.)

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 $w \; Rv$, 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 **STS**es in representation invariants per state.
- Bisimilar states can have different representation invariants.
- 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 $\boxed{s; \mathsf{T}}^{\gamma}$

STSAlloc

$$\frac{}{\varphi(s) \Rrightarrow \exists \gamma . \boxed{\varphi}^{\gamma}_{\pi} * \boxed{s; \mathsf{AllTokens} \setminus \mathcal{T}(s)}^{\gamma}}$$

STSOpen

$$\frac{}{\boxed{\varphi}^{\gamma}_{\pi} * \boxed{s; \mathsf{T}}^{\gamma} \Rrightarrow (\exists\, s' . \ulcorner(\mathsf{s0}, \mathsf{T}) \overset{\mathsf{rely}^*}{\sqsubseteq}_{\pi} (\mathsf{s}', \mathsf{T})\urcorner * \varphi(s) * \forall \mathsf{sl}',\ \mathsf{T}' . \ulcorner(\mathsf{s}', \mathsf{T}) \overset{\mathsf{guar.}^*}{\sqsubseteq}_{\pi} (\mathsf{sl}', \mathsf{T}')\urcorner * \varphi(\mathsf{sl}') \Rrightarrow \boxed{\mathsf{sl}; \mathsf{T}'}^{\gamma})}$$

UpdIsl

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

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

Decomposing Bisimilarity in STSes

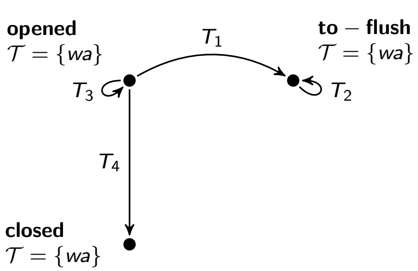
The bisimulation ($\mathcal{M}(\pi, \pi', \varphi, \varphi', s, \mathrm{T}, \mathrm{U})$) between two state machines, π and π' is composed of

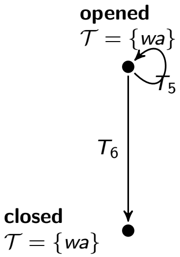
- The source STS – π
- The target STS – π'
- The source STS's state interpretation function – φ
- The target STS's state interpretation function – φ'
- Token Embedding – $\epsilon_{\mathcal{S}} : \ \mathcal{S}(\pi) \mapsto \mathcal{S}(\pi')$
- State Embedding – $\epsilon_{\mathcal{T}} : \ \mathcal{T}(\pi) \mapsto \mathcal{T}(\pi')$
- 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!

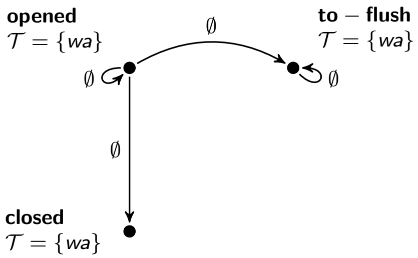
Embeddings for Tokens

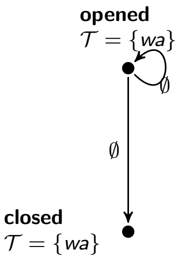




- How to impose *indistinguishability* of behavior observed on *bisimilar* states
- How to impose *indistinguishability* over the steps taken between *bisimilar* states
- Make *token exchange* align with *indistinguishability*: Embedding tokens (lifted $\epsilon_{\overline{\mathcal{T}}}$) T_1, T_3, and T_2 to T_5, and T_4 to T_6

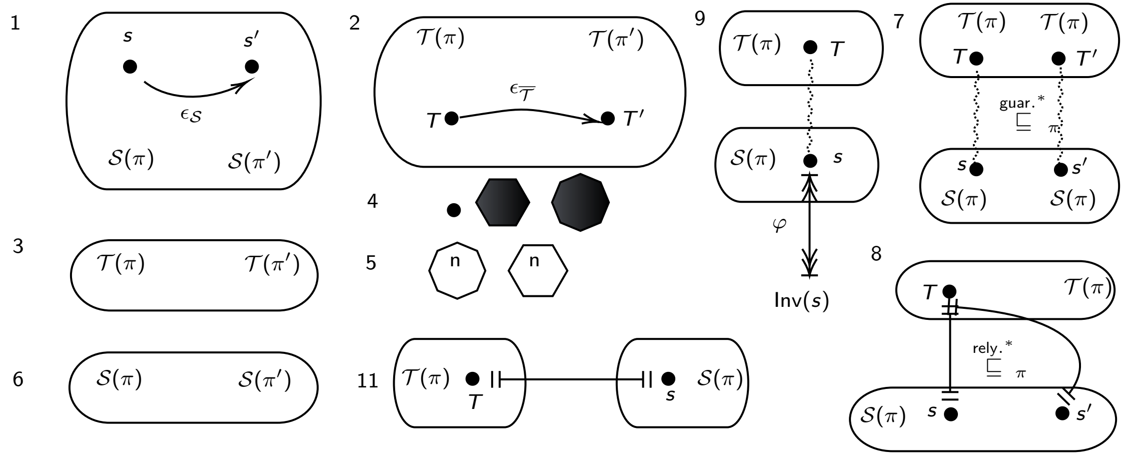
Embeddings for State and Tokens





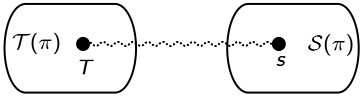
- Setting client interference aside, we would consider opened and flushed of the left state machine to be *bisimilar* to opened of the right state machine

A Quick Tour on the Represention of Laws - 1

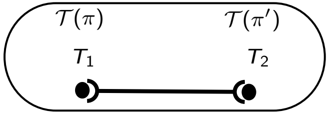


A Quick Tour on the Represention of Laws - 2

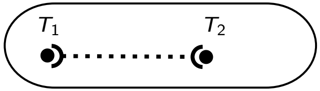
12



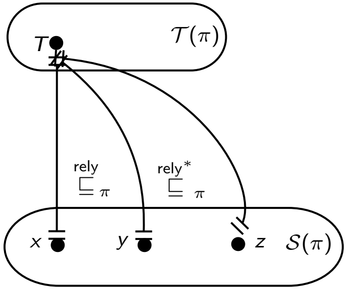
13



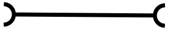
10



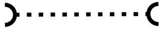
14



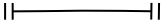
15



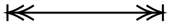
16



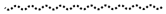
18



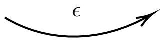
19



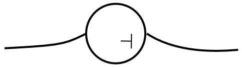
20



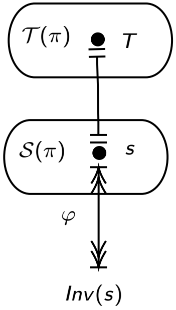
21



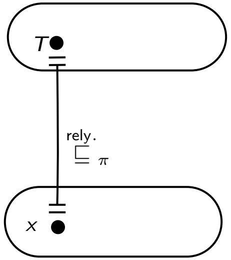
22



23



17



We Use This

UpdIsl

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

We Need This

UpdIsl

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

Bisim

$$\frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s; \epsilon_{\mathcal{T}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

How? Obligation - 1

UpdIsl

$$\frac{\alpha \text{ physically atomic} \qquad \forall s_0 \,.\, \left((s;\mathsf{T}) \overset{\mathsf{rely}^*}{\sqsubseteq}_{\pi} (s_0;\mathsf{T})\right) \vdash \{\varphi(s_0) * P\}\ \alpha\ \{\exists s',\ \mathsf{T}' \,.\, (s_0;\mathsf{T}) \overset{\mathsf{guar}^*}{\sqsubseteq}_{\pi} (s';\mathsf{T}') * \varphi(s') * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{s;\mathsf{T}}^{\gamma}_{\pi} * P\}\ \alpha\ \{\exists\, s',\mathsf{T}'.\boxed{s';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

Bisim

$$\frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s;\epsilon_{\overline{\mathcal{T}}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

The Law of Rely

Theorem (The Law of Rely)

$$\forall s'.(s;\, T) \overset{\text{rely}^*}{\sqsubseteq}_{\pi} (s';\, T) \leftrightarrow$$
$$(\forall_{s_1, s_1', T_1} . \epsilon_{\mathcal{S}}(s, s_1) \rightarrow \epsilon_{\mathcal{S}}(s', s_1') \rightarrow \epsilon_{\overline{\mathcal{T}}}(T, T_1) \rightarrow (s_1;\, T_1) \overset{\text{rely}^*}{\sqsubseteq}_{\pi'} (s_1';\, T_1)$$

- 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

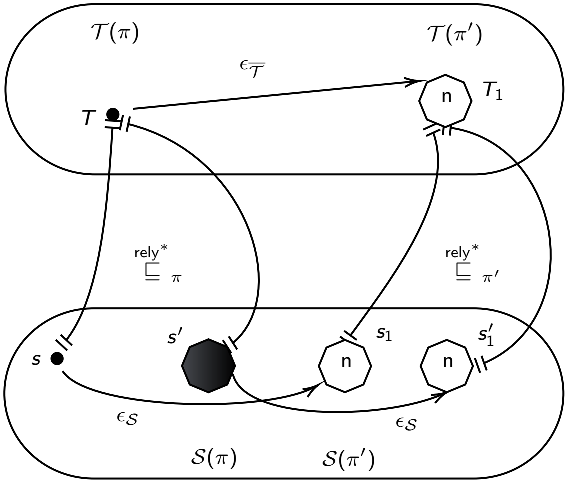
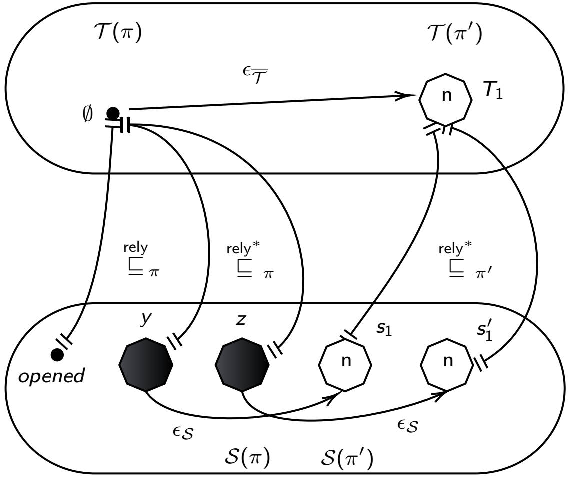
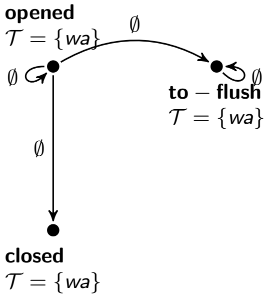
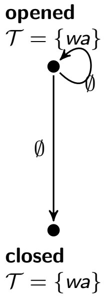


Figure: Embedding The Client Interface

The Law of Rely for a Bisimulation Instance







We Need This

UpdIsl

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

Bisim

$$\frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s;\epsilon_{\mathcal{T}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

How? Obligation - 2

UPDISL

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

BISIM

$$\frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s;\epsilon_{\overline{\mathcal{T}}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

The Law of Guarantee

Theorem (Guarantee Bisim without Invariants)

$$
\begin{aligned}
&\forall_{q',q,T'}.\ \epsilon_{\overline{\mathcal{T}}}(T) \equiv T' \to \epsilon_{\mathcal{S}}(s,q) \to (q;T') \overset{\text{rely}^*}{\sqsubseteq}_{\pi'} (q';T') \to \\
&\quad \forall_{q'',T''}.\ (q';T') \overset{\text{guar.}}{\sqsubseteq}_{\pi'} (q'';T'') \to \\
&\quad\quad \exists_{s',s'',T_0',T_0''}.\ (s';T_0') \overset{\text{guar.}}{\sqsubseteq}_{\pi} (s'';T_0'') \wedge \\
&\quad\quad\quad \epsilon_{\mathcal{S}}(s') = q' \wedge \epsilon_{\mathcal{S}}(s'') = q'' \wedge \epsilon_{\overline{\mathcal{T}}}(T_0') \equiv T' \wedge \epsilon_{\overline{\mathcal{T}}}(T_0'') \equiv T''
\end{aligned}
$$

- *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

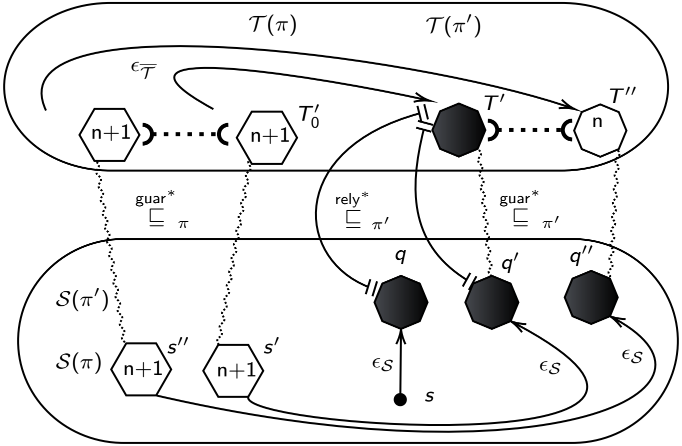
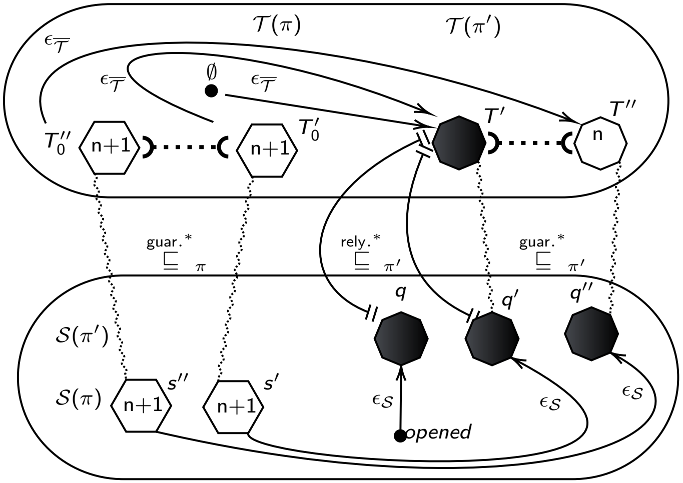
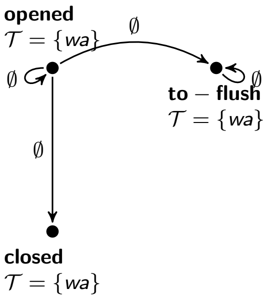
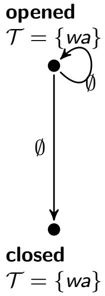


Figure: Embedding the Guarantee Steps of Target STS

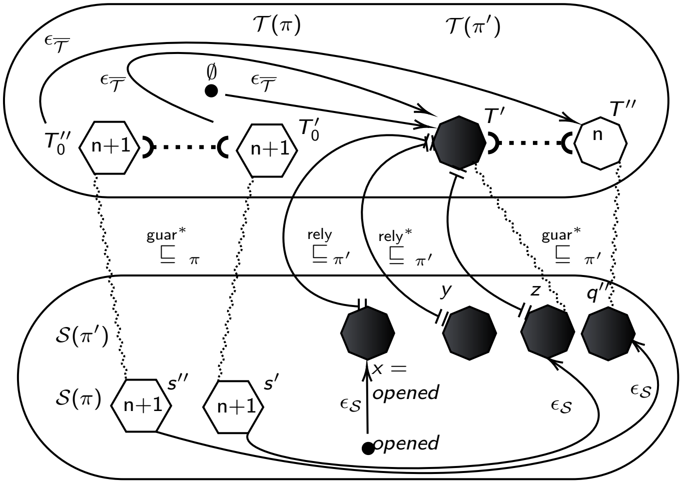
The Law of Guarantee for the Bisimulation Instance

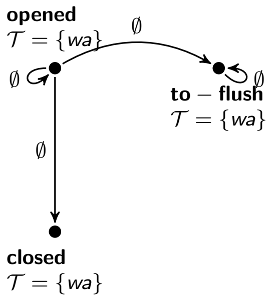


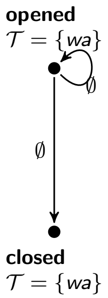




The Law of Guarantee for the Bisimulation Instance







We Need This

$$\textsc{UpdIsl}\quad \frac{\alpha \text{ physically atomic} \qquad \forall s_0 \,.\, \left((s;\mathsf{T}) \overset{\mathsf{rely}^*}{\sqsubseteq}_{\pi} (s_0;\mathsf{T})\right) \vdash \{\varphi(s_0) * P\}\ \alpha\ \{\exists s',\mathsf{T}' \,.\, (s_0;\mathsf{T}) \overset{\mathsf{guar}^*}{\sqsubseteq}_{\pi} (s';\mathsf{T}') * \varphi(s') * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{s;\mathsf{T}}^{\gamma}_{\pi} * P\}\ \alpha\ \{\exists\, s',\mathsf{T}' .\, \boxed{s';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

$$\textsc{Bisim}\quad \frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s;\epsilon_{\mathcal{T}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

How? Obligation - 3

UpdIsl

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

Bisim

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

How? Obligation - 4

UpdIsl

$$\frac{\alpha \text{ physically atomic} \qquad \forall s_0 \,.\, \left((s;\mathsf{T}) \mathrel{\overset{\mathsf{rely}^*}{\sqsubseteq}_{\pi}} (s_0;\mathsf{T})\right) \vdash \{\boxed{\varphi(s_0)} * P\}\ \alpha\ \{\exists s', \mathsf{T}' \,.\, (s_0;\mathsf{T}) \mathrel{\overset{\mathsf{guar}^*}{\sqsubseteq}_{\pi}} (s';\mathsf{T}') * \boxed{\varphi(s')} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{s;\mathsf{T}}^{\gamma}_{\pi} * P\}\ \alpha\ \{\exists\, s', \mathsf{T}' .\, \boxed{s';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

Bisim

$$\frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s;\epsilon_{\overline{\mathcal{T}}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

How? Intuition on Invariants in Bisimulation

$$\textsc{UpdIsl}\quad \frac{\alpha \text{ physically atomic} \qquad \forall s_0 \,.\, ((s;\mathsf{T}) \overset{\mathsf{rely}^*}{\sqsubseteq}_\pi (s_0;\mathsf{T})) \vdash \{\boxed{\varphi(s_0)} * P\}\ \alpha\ \{\exists s',\ \mathsf{T}' \,.\, (s_0;\mathsf{T}) \overset{\mathsf{guar}^*}{\sqsubseteq}_\pi (s';\mathsf{T}') * \boxed{\varphi(s')} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{s;\mathsf{T}}^{\gamma} * P\}\ \alpha\ \{\exists\, s',\mathsf{T}' . \boxed{s';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

$$\textsc{Bisim}\quad \frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s;\epsilon_{\mathcal{T}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

- 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
  - $\varphi'(\epsilon_{\mathcal{S}}(s0)) \vdash \exists\, s' . \varphi(s')$
  - and precondition in the source STS – $\varphi(s0)$
- To obtain $\exists\, s' \,.\, \varphi(s')$

Invariants of File Protocols

Definition (File Protocol Invariants)

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

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

The Law of Tolerance

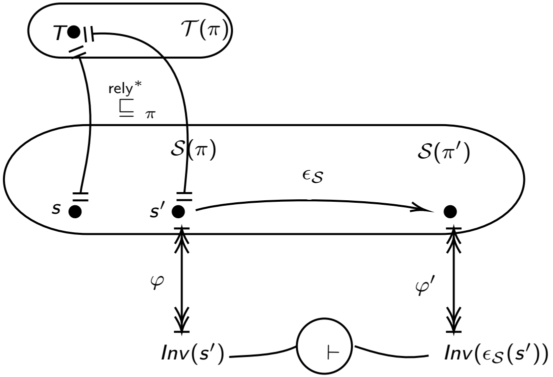


Figure: The Law of Tolerance for a Valid Pre-Condition

Theorem (The Law of Tolerance)

$$\forall\ s'\ .\ (s\ \mathrm{T})\ \overset{\text{rely}^*}{\sqsubseteq}_\pi\ (s';\mathrm{T}) \leftrightarrow \boxed{\varphi(s') \vdash \varphi'(\epsilon_\mathcal{S}(s'))}$$

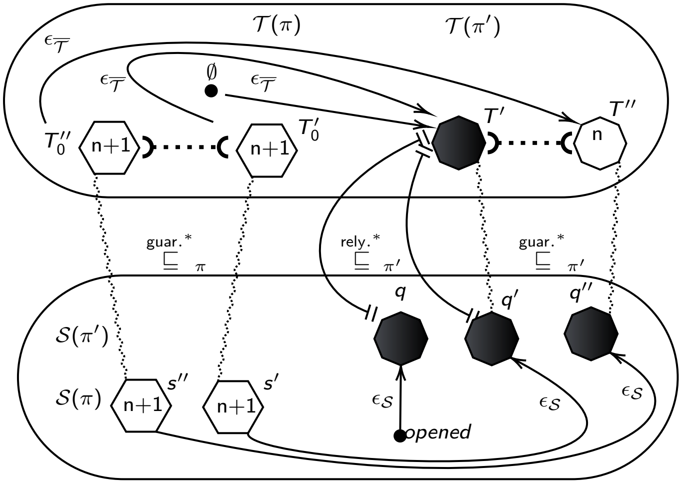
- The source STS's precondition for the embedded states is stronger
- φ(flushed) of π or φ(opened) of π implies φ'(opened) of π'

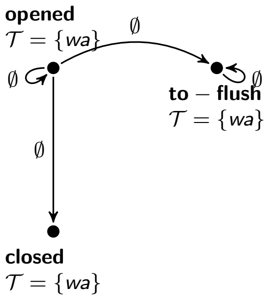
Revisiting the Law of Guarantee for Invariants

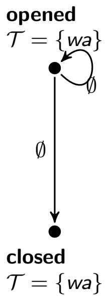
Theorem (The Law of Guarantee with Invariants)

$$\begin{aligned}
&\forall\, q' \;.\; (\epsilon_{\mathcal{S}}(s); \epsilon_{\overline{\mathcal{T}}}(\mathrm{T})) \overset{\text{rely}^*}{\sqsubseteq}_{\pi'} (\epsilon_{\mathcal{S}}(q'); \epsilon_{\overline{\mathcal{T}}}(\mathrm{T})) \rightarrow \\
&\quad \forall\, q'',\ \mathrm{T}'' \;.\; (q'; \epsilon_{\overline{\mathcal{T}}}(\mathrm{T})) \overset{\text{guar}^*}{\sqsubseteq}_{\pi'} (q''; \mathrm{T}'') \rightarrow \\
&\qquad \exists\, s'\ s''\ \mathrm{T0}'\ \mathrm{T0}'' \;.\; (s'; \mathrm{T0}') \overset{\text{guar}^*}{\sqsubseteq}_{\pi} (s''; \mathrm{T0}'') \land \\
&\qquad\quad \epsilon_{\mathcal{S}}(s') = q' \land \epsilon_{\mathcal{S}}(s'') = q'' \land \\
&\qquad\quad (\epsilon_{\overline{\mathcal{T}}}(\mathrm{T0}')) = (\epsilon_{\overline{\mathcal{T}}}(\mathrm{T})) \;\land\; (\epsilon_{\mathcal{T}}(\mathrm{T0}')) = \mathrm{T}'' \;\land \\
&\qquad\quad \boxed{\varphi'(q'') \vdash \varphi(s'')}
\end{aligned}$$

Revisiting the Law of Guarantee for Invariants





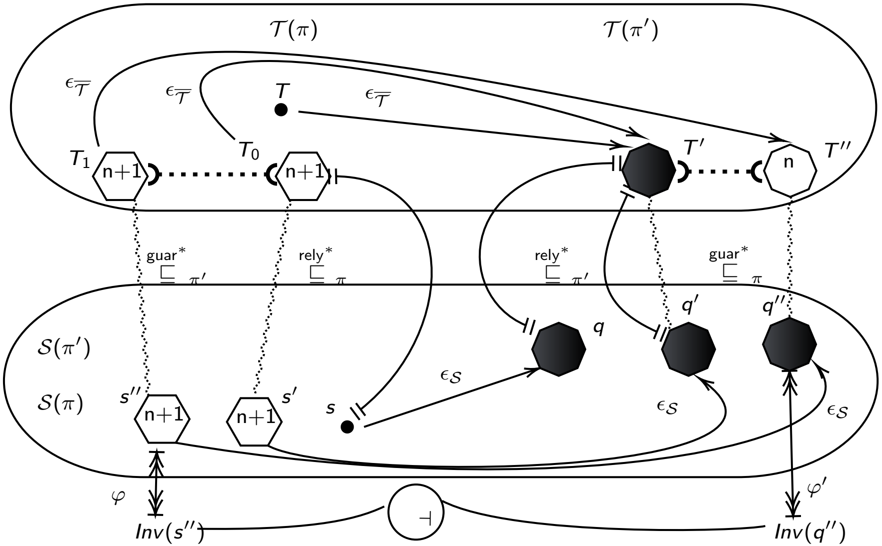


Soundness

Theorem (Soundness)

The updated state from UPDISL *is preserved by the bisimulation.*

Soundness



$\mathcal{T}(\pi)$
$\mathcal{T}(\pi')$
$\epsilon_{\mathcal{T}}$
$\epsilon_{\mathcal{T}}$
T
$\epsilon_{\mathcal{T}}$
T_1
n+1
T_0
n+1
T'
n
T''
guar* $\sqsubseteq_{\pi'}$
rely* $\sqsubseteq_{\pi}$
rely* $\sqsubseteq_{\pi'}$
guar* $\sqsubseteq_{\pi}$
q'
q''
q
$\mathcal{S}(\pi')$
$\mathcal{S}(\pi)$
s''
n+1
n+1
s'
s
$\epsilon_{\mathcal{S}}$
$\epsilon_{\mathcal{S}}$
$\epsilon_{\mathcal{S}}$
φ
φ'
$\dashv$
$Inv(s'')$
$Inv(q'')$

Keeping Promises

$$\dfrac{\pi \sim \pi' \qquad \text{opened} \; \epsilon_{\mathcal{S}} \; s \qquad q' \; \epsilon_{\mathcal{S}} \; s' \\ \{ \boxed{s; \epsilon_{\overline{\mathcal{T}}}(\mathcal{T})}^{\gamma}_{\pi'} * P \} \; \text{write} \; \ell \; \text{new\_val} \; \{ \boxed{s'; \mathsf{T}'}^{\gamma}_{\pi'} * Q \}}{\boxed{\varphi_{\text{distributedfile}}}^{\gamma}_{\pi} \vdash \{ \boxed{\text{opened}; \mathsf{T}}^{\gamma}_{\pi} * P \} \; C \; \{ \boxed{q'; \mathsf{T}'}^{\gamma}_{\pi} * Q \}} \; \textsc{Transfer File Write}$$

On-going 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 $\textsc{HeapLang}$
- 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?

References

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.
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*

The End