Detecting Inconsistencies in Arm CCA's Formally Verified Specification

Changho Choi (Samsung Research),
Xiang Cheng (Georgia Institute of Technology),
Bokdeuk Jeong (Samsung Research),
Taesoo Kim (Samsung Research, Georgia Institute of Technology)

The Critical Role of Specifications

Formal verficiation is only as good as its specification.

- High-Assurance Systems: Formal verification provides mathematitcal rigor for kernels and security-critical systems.

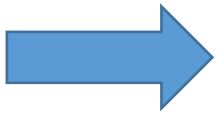
- Specification as a TCB: Bugs in specs can invalidate proofs and introduce critical vulnerabilities.

The Critical Role of Specifications

Formal verficiation is only as good as its specification.

- High-Assurance Systems: Formal verification provides mathematitcal rigor for kernels and security-critical systems.

- Specification as a TCB: Bugs in specs can invalidate proofs and introduce critical vulnerabilities.



Enhancing the trustworthiness of specifications is important.

Previous approaches

- Formally proving security properties (OSDI22, OOPSLA23).
- Specification auditing (EuroSys17, OOPSLA17).
- Testing specifications against implementations (ITP10, CAV16).

Previous approaches

Hard to scale

- Formally proving security properties (OSDI22, OOPSLA23).
- Specification auditing (EuroSys17, OOPSLA17).
- Testing specifications against implementations (ITP10, CAV16).

Previous approaches

Hard to scale

- Formally proving security properties (OSDI22, OOPSLA23).
- Specification auditing (EuroSys17, OOPSLA17).

- Testing specifications against implementations (ITP10, CAV16).

Requires implementation

Our approach (Scope)

If two related statements in the same document contradict, at least one is incorrect.

- An automated system that identifies specification inconsistencies by operating solely on specification documents.
- Scope translates the specification into a machine-verifiable model using Verus and SMT solvers, then detects inconsistencies.

The RMM specification

- Confidential Compute Architecture (CCA) enables hardware-enforced isolation for confidential computing workloads.

- The RMM specification describes the functionalities of the Realm Management Monitor (RMM), privileged firmware.

- The security guarantees of the entire CCA depend on this document.
  - Arm CCA is expected to cope with rising computations in data centers, superseding its predecessor called TrustZone.

Workflow

Formal reasoning

Rule-based checks

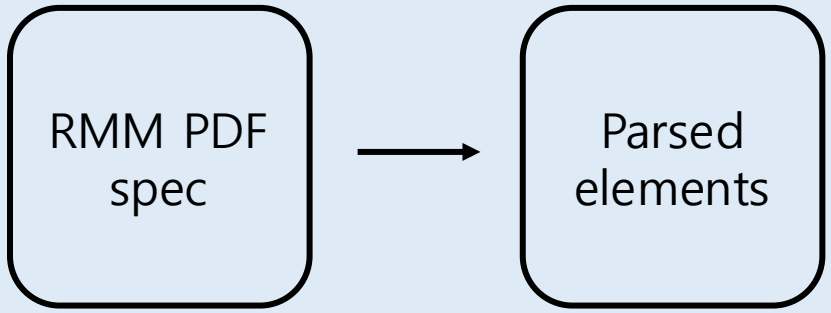
Workflow

Formal reasoning

Rule-based checks

Workflow

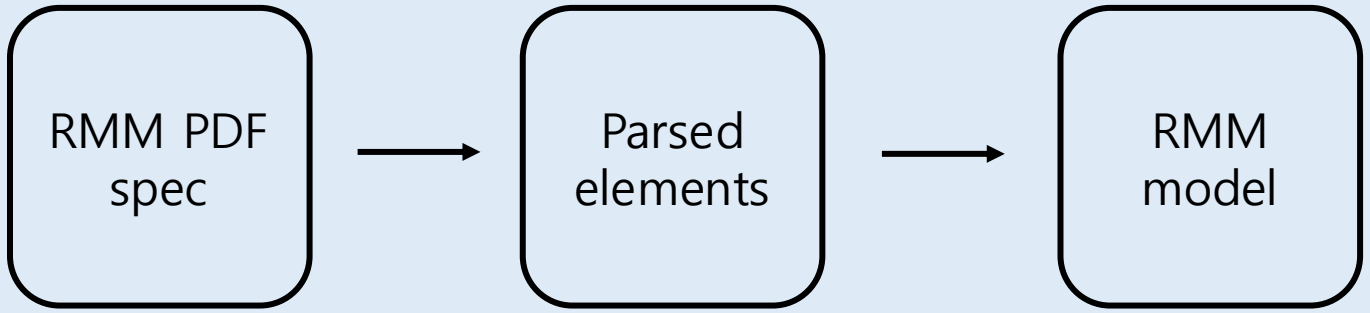
Formal reasoning



Rule-based checks

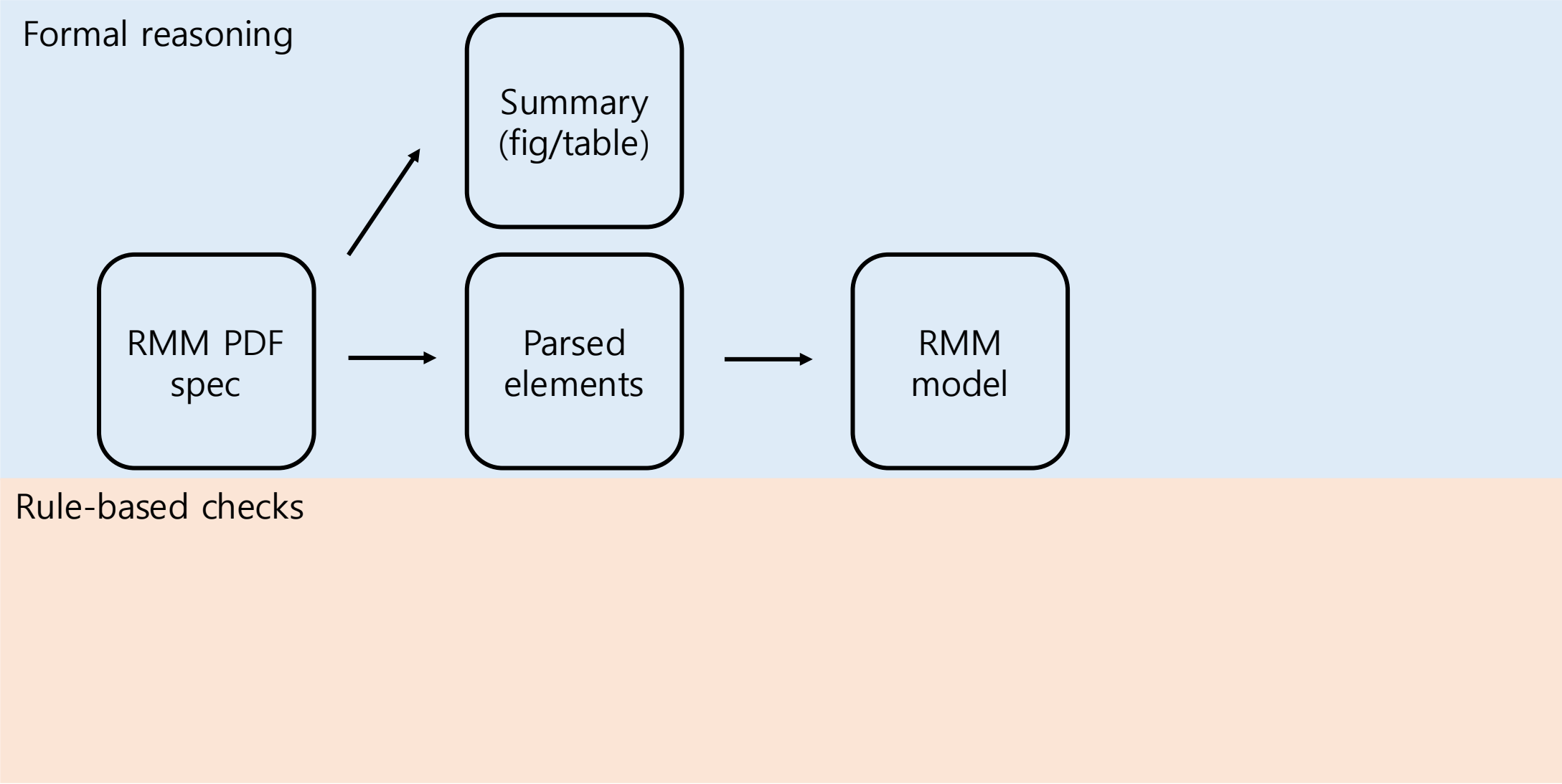
Workflow

Formal reasoning

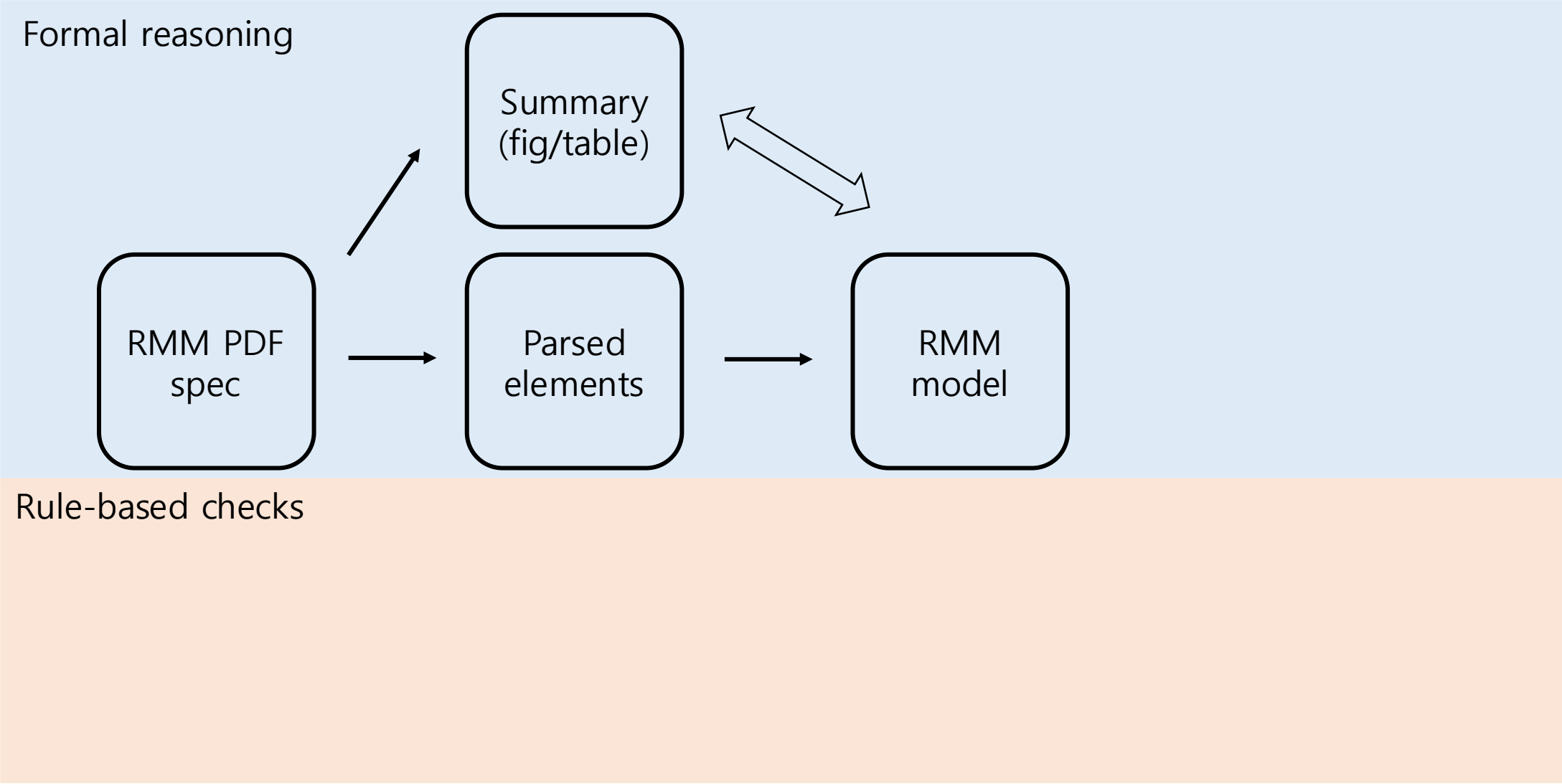


Rule-based checks

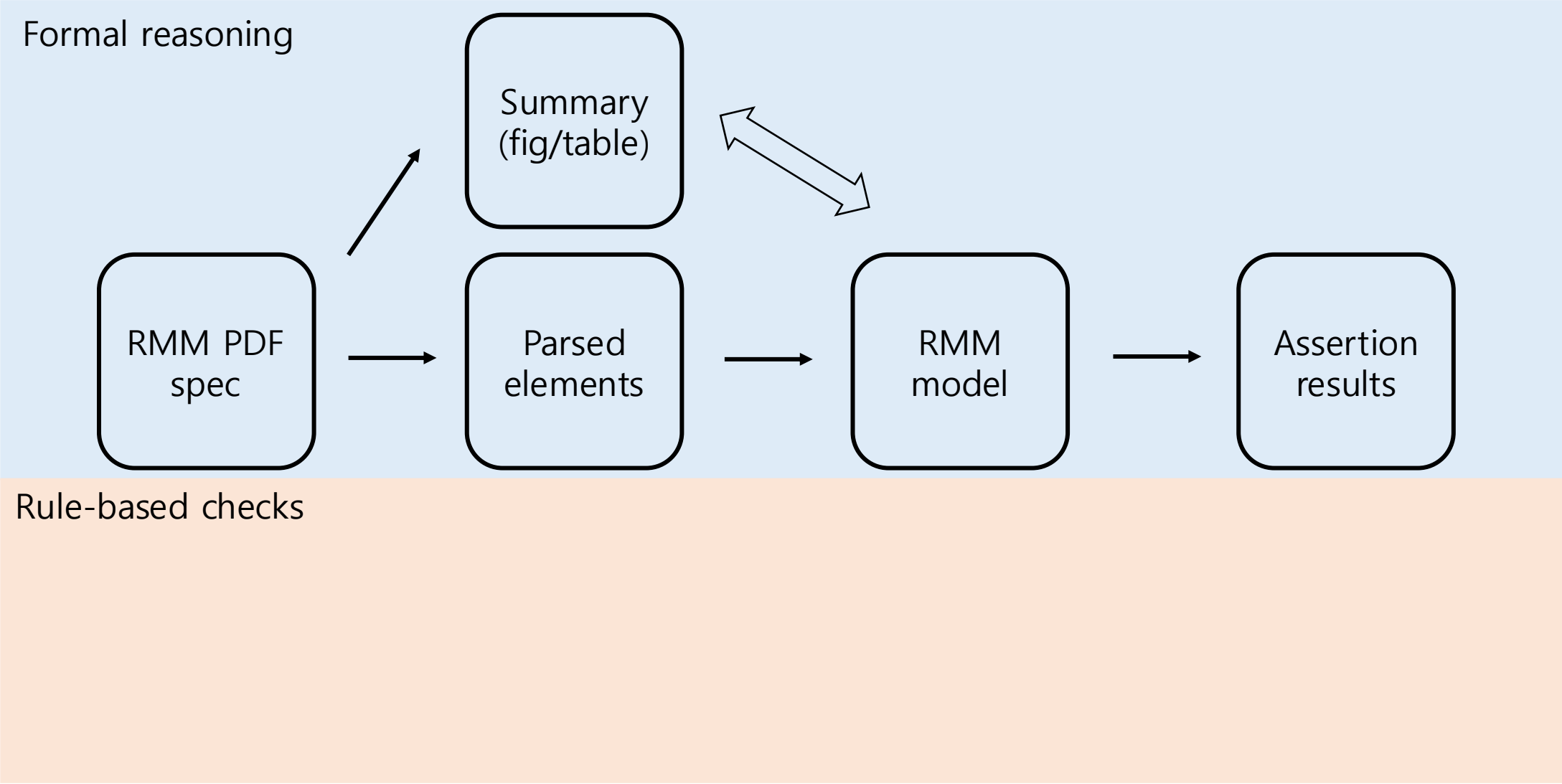
Workflow


Formal reasoning
Summary (fig/table)
RMM PDF spec
Parsed elements
RMM model
Rule-based checks

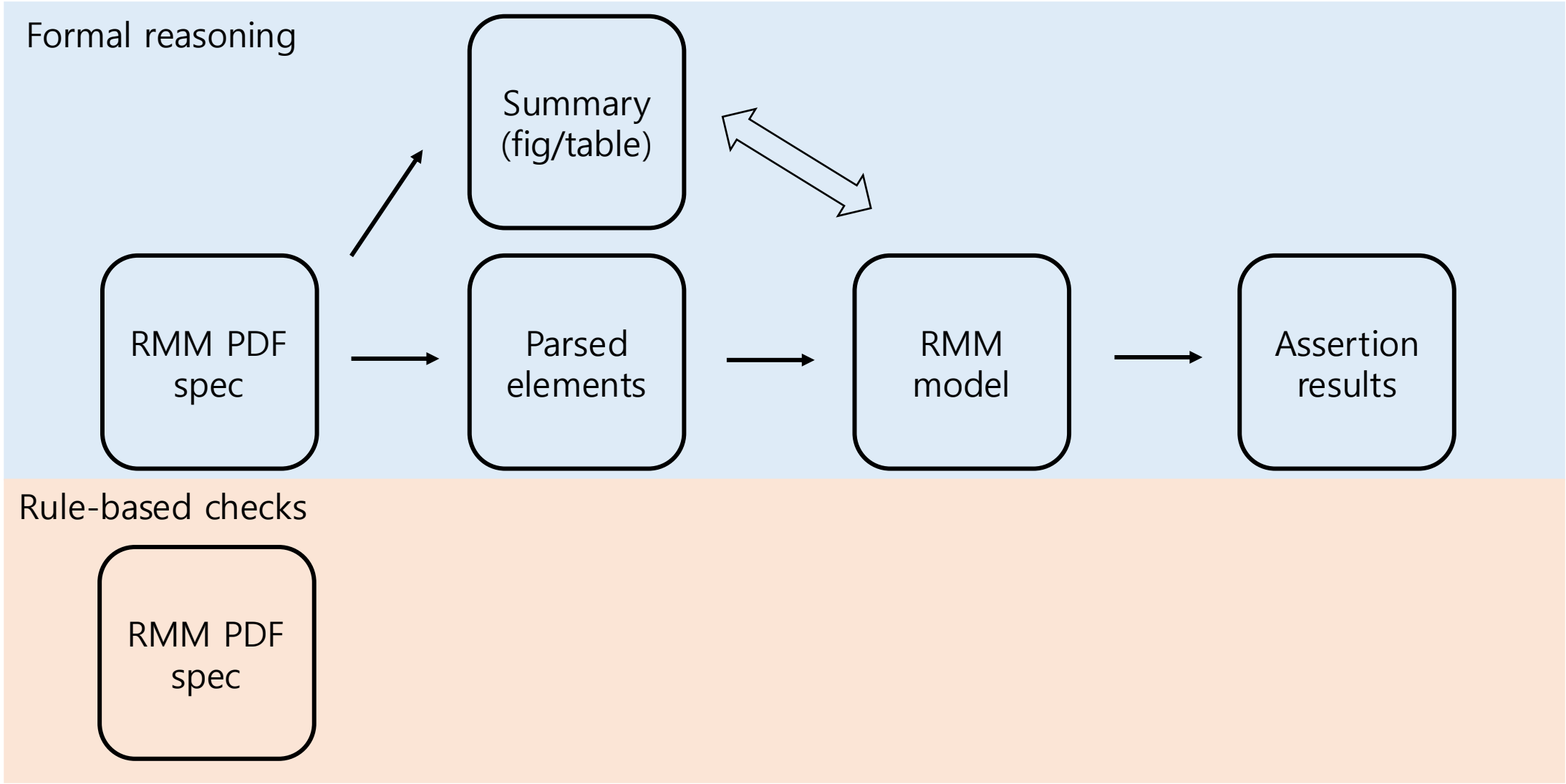
Workflow


Formal reasoning
Summary (fig/table)
RMM PDF spec
Parsed elements
RMM model
Rule-based checks

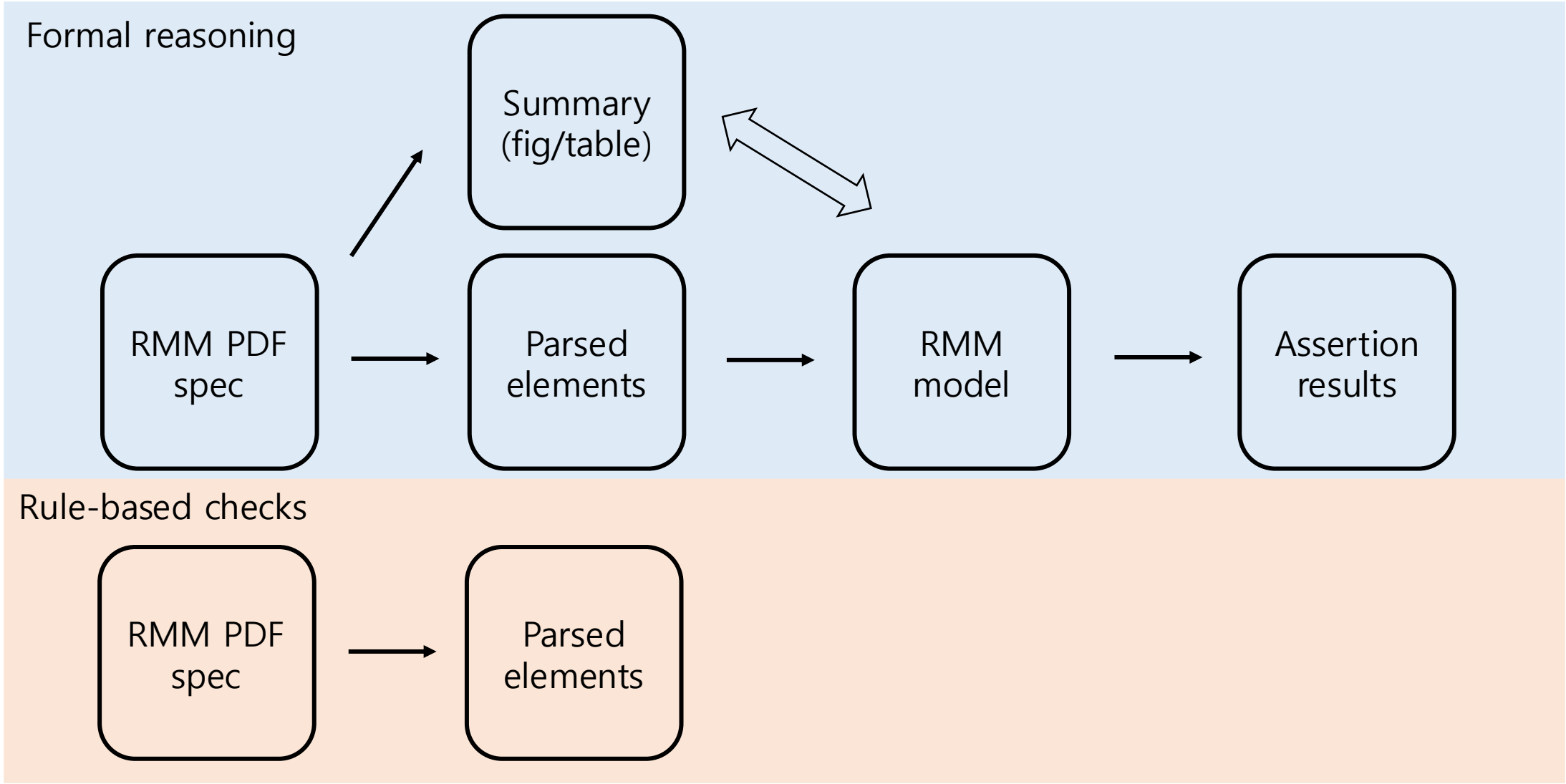
Workflow


Formal reasoning
Summary (fig/table)
RMM PDF spec
Parsed elements
RMM model
Assertion results
Rule-based checks

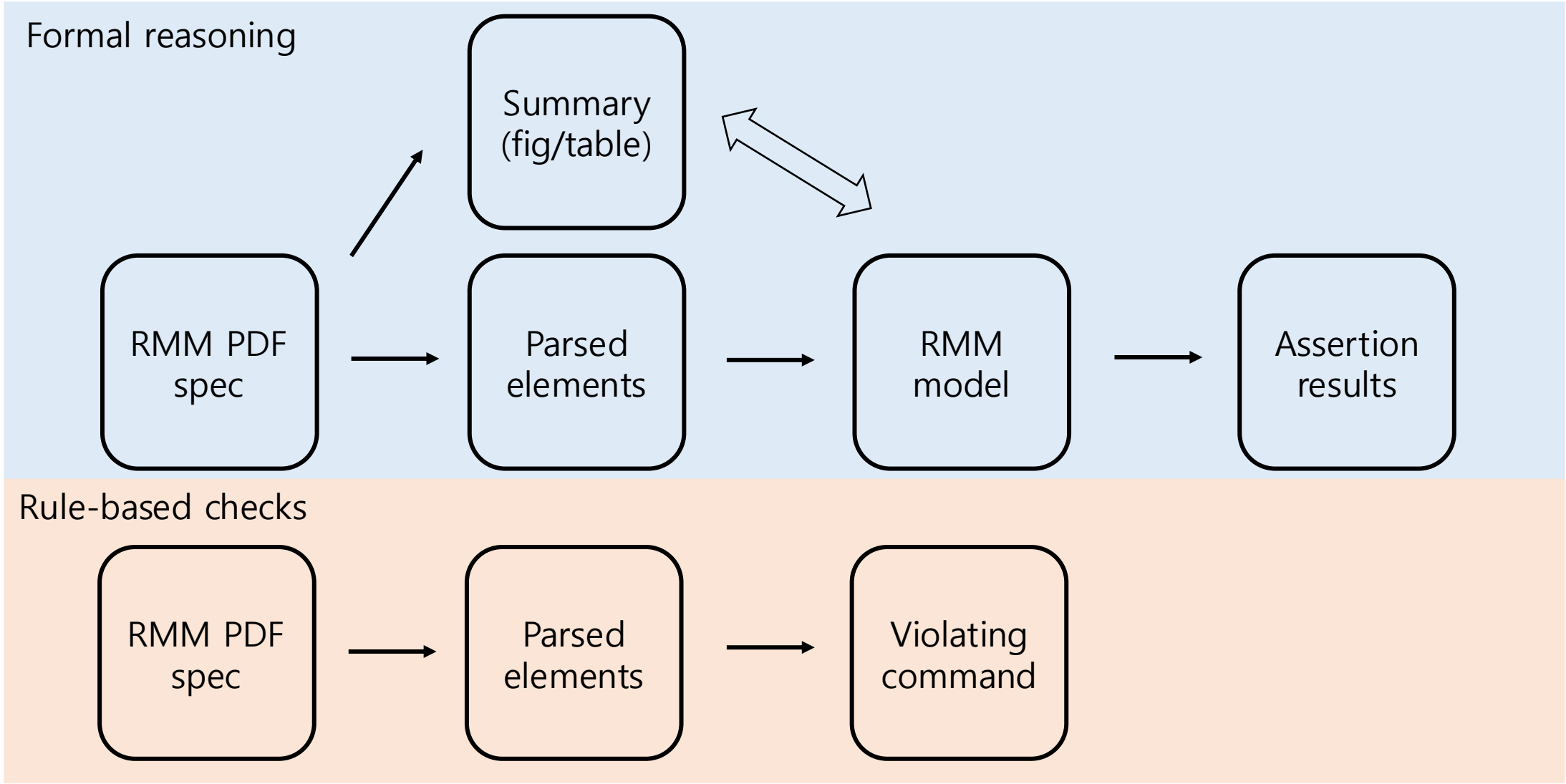
Workflow


Formal reasoning
Summary (fig/table)
RMM PDF spec
Parsed elements
RMM model
Assertion results
Rule-based checks
RMM PDF spec

Workflow


Formal reasoning
Summary (fig/table)
RMM PDF spec
Parsed elements
RMM model
Assertion results
Rule-based checks
RMM PDF spec
Parsed elements

Workflow


Formal reasoning
Summary (fig/table)
RMM PDF spec
Parsed elements
RMM model
Assertion results
Rule-based checks
RMM PDF spec
Parsed elements
Violating command

Formal reasoning

assume(RMM model) assert(RMM summary)

Formal reasoning

assume(RMM model)

…

RMI DATA DESTROY spec

assert(RMM summary)

…

RMI DATA DESROY summary

Formal reasoning

assume(RMM model)

…

RMI DATA DESTROY spec

…

RIPAS Dependency spec

(ANY)

assert(RMM summary)

…

RMI DATA DESROY summary

…

RIPAS Dependency summary

(RAM or EMPTY)

Formal reasoning

assume(RMM model)	assert(RMM summary)
…	…
RMI DATA DESTROY spec	RMI DATA DESROY summary
…	…
RIPAS Dependency spec (ANY)	RIPAS Dependency summary (RAM or EMPTY)

An assertion violation occurs,
since RIPAS can be DESTROYED

Command conversion

B4.3.3.1 Interface

B4.3.3.1.1 Input values

Name	Reg.	Bits	Type
fid	X0	63:0	UInt64
rd	X1	63:0	Address
ipa	X2	63:0	Address

B4.3.3.1.3 Output values

Name	Reg.	Bits	Type
result	X0	63:0	CommandReturnCode
data	X1	63:0	Address
top	X2	63:0	Address

B4.3.3.1.2 Context

Name	Type	Value	Before
walk	RttWalkResult	RttWalk(rd, ipa, RMM_RTT_PAGE_LEVEL)	false
walk_top	Address	RttSkipNonLiveEntries(Rtt(walk.rtt_addr), walk.level, ipa)	false

B4.3.3.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_state	pre: walk.rtte.state != ASSIGNED post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))

B4.3.3.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED
ripas_ram	pre: walk.rtte.ripas == RAM post: walk.rtte.ripas == DESTROYED

```
pub open spec fn rmi_data_destroy_spec(rd: Address,
  ipa: Address, res: Result<(), RmiStatusCode>,
  data: Address, top: Address, old_s: S, new_s: S) -> bool
{
  // Failure conditions
  (!AddrIsGranuleAligned(old_s, rd)
    ==> ResultEqual(res, RMI_ERROR_INPUT))
  ...
  && (old_walk.rtte.state != ASSIGNED
    ==> (ResultEqual(res, RMI_ERROR_RTT(new_walk.level))
      && (top == RttSkipNonLiveEntries(new_s, Rtt(new_s,
          new_walk.rtt_addr), new_walk.level, ipa))))
  // Success conditions
  ...
  && (res.is_Ok()
    ==> new_walk.rtte.state == UNASSIGNED)
  && (res.is_Ok() && old_walk.rtte.ripas == RAM
    ==> new_walk.rtte.ripas == DESTROYED)
  ...
}
```

RMI_DATA_DESTROY, From the RMM specification (1.0-eac5).

Command conversion

B4.3.3.1 Interface

B4.3.3.1.1 Input values				B4.3.3.1.3 Output values			
Name	Reg.	Bits	Type	Name	Reg.	Bits	Type
fid	X0	63:0	UInt64	result	X0	63:0	CommandReturnCode
rd	X1	63:0	Address	data	X1	63:0	Address
ipa	X2	63:0	Address	top	X2	63:0	Address

B4.3.3.1.2 Context

Name	Type	Value	Before
walk	RttWalkResult	RttWalk(rd, ipa, RMM_RTT_PAGE_LEVEL)	false
walk_top	Address	RttSkipNonLiveEntries(Rtt(walk.rtt_addr), walk.level, ipa)	false

B4.3.3.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_state	pre: walk.rtte.state != ASSIGNED post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))

B4.3.3.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED
ripas_ram	pre: walk.rtte.ripas == RAM post: walk.rtte.ripas == DESTROYED

```
pub open spec fn rmi_data_destroy_spec(rd: Address,
  ipa: Address, res: Result<(), RmiStatusCode>,
  data: Address, top: Address, old_s: S, new_s: S) -> bool
{
  // Failure conditions
  (!AddrIsGranuleAligned(old_s, rd)
    ==> ResultEqual(res, RMI_ERROR_INPUT))
  ...
  && (old_walk.rtte.state != ASSIGNED
    ==> (ResultEqual(res, RMI_ERROR_RTT(new_walk.level))
      && (top == RttSkipNonLiveEntries(new_s, Rtt(new_s,
          new_walk.rtt_addr), new_walk.level, ipa))))
  // Success conditions
  ...
  && (res.is_Ok()
    ==> new_walk.rtte.state == UNASSIGNED)
  && (res.is_Ok() && old_walk.rtte.ripas == RAM
    ==> new_walk.rtte.ripas == DESTROYED)
  ...
}
```

RMI_DATA_DESTROY, From the RMM specification (1.0-eac5).

Command conversion

B4.3.3.1 Interface

B4.3.3.1.1 Input values

Name	Reg.	Bits	Type
fid	X0	63:0	UInt64
rd	X1	63:0	Address
ipa	X2	63:0	Address

B4.3.3.1.3 Output values

Name	Reg.	Bits	Type
result	X0	63:0	CommandReturnCode
data	X1	63:0	Address
top	X2	63:0	Address

B4.3.3.1.2 Context

Name	Type	Value	Before
walk	RttWalkResult	RttWalk(rd, ipa, RMM_RTT_PAGE_LEVEL)	false
walk_top	Address	RttSkipNonLiveEntries(Rtt(walk.rtt_addr), walk.level, ipa)	false

B4.3.3.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_state	pre: walk.rtte.state != ASSIGNED post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))

B4.3.3.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED
ripas_ram	pre: walk.rtte.ripas == RAM post: walk.rtte.ripas == DESTROYED

```
pub open spec fn rmi_data_destroy_spec(rd: Address,
  ipa: Address, res: Result<(), RmiStatusCode>,
  data: Address, top: Address, old_s: S, new_s: S) -> bool
{
  // Failure conditions
  (!AddrIsGranuleAligned(old_s, rd)
    ==> ResultEqual(res, RMI_ERROR_INPUT))
  ...
  && (old_walk.rtte.state != ASSIGNED
    ==> (ResultEqual(res, RMI_ERROR_RTT(new_walk.level))
      && (top == RttSkipNonLiveEntries(new_s, Rtt(new_s,
          new_walk.rtt_addr), new_walk.level, ipa))))
  // Success conditions
  ...
  && (res.is_Ok()
    ==> new_walk.rtte.state == UNASSIGNED)
  && (res.is_Ok() && old_walk.rtte.ripas == RAM
    ==> new_walk.rtte.ripas == DESTROYED)
  ...
}
```

RMI_DATA_DESTROY, From the RMM specification (1.0-eac5).

Command conversion

B4.3.3.1 Interface

B4.3.3.1.1 Input values

Name	Reg.	Bits	Type
fid	X0	63:0	UInt64
rd	X1	63:0	Address
ipa	X2	63:0	Address

B4.3.3.1.3 Output values

Name	Reg.	Bits	Type
result	X0	63:0	CommandReturnCode
data	X1	63:0	Address
top	X2	63:0	Address

B4.3.3.1.2 Context

Name	Type	Value	Before
walk	RttWalkResult	RttWalk(rd, ipa, RMM_RTT_PAGE_LEVEL)	false
walk_top	Address	RttSkipNonLiveEntries(Rtt(walk.rtt_addr), walk.level, ipa)	false

B4.3.3.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_state	pre: walk.rtte.state != ASSIGNED post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))

B4.3.3.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED
ripas_ram	pre: walk.rtte.ripas == RAM post: walk.rtte.ripas == DESTROYED

```
pub open spec fn rmi_data_destroy_spec(rd: Address,
  ipa: Address, res: Result<(), RmiStatusCode>,
  data: Address, top: Address, old_s: S, new_s: S) -> bool
{
   ...
   // A deduced condition from failure conditions
   && ((AddrIsGranuleAligned(old_s, rd) &&
        ...
        !(old_walk.rtte.state != ASSIGNED))
     ==> res.is_Ok())
   // Deduced conditions from success conditions
   ...
   && (res.is_Err()
     ==> new_walk.rtte.state == old_walk.rtte.state)
   && (!(res.is_Ok() && (old_walk.rtte.ripas == RAM))
     ==> new_walk.rtte.ripas == old_walk.rtte.ripas)
}
```

RMI_DATA_DESTROY, From the RMM specification (1.0-eac5).

Summary conversion

RMI command	Dependency on RIPAS	Dependency on HIPAS	New RIPAS	New HIPAS
DATA_DESTROY	EMPTY	ASSIGNED	Unchanged	UNASSIGNED
DATA_DESTROY	RAM	ASSIGNED	DESTROYED	UNASSIGNED

Dependency of RMI command execution,
From the RMM specification (1.0-eac5).

```
pub proof fn rmi_data_destroy_rule (rd: Address,
  ipa: Address, res: Result<(), RmiStatusCode>,
  data: Address, top: Address, old_s: S, new_s: S)
    requires rmi_data_destroy_spec(rd, ipa, res,
                                   data, top, old_s, new_s),
{
  let old_walk = RttWalk_(old_s, rd, ipa,
                          RMM_RTT_PAGE_LEVEL);
  let new_walk = RttWalk_(new_s, rd, ipa,
                          RMM_RTT_PAGE_LEVEL);
  // Dependency on RIPAS (If RIPAS is EMPTY or RAM)
  // The below would trigger an assertion violation
  assert(res.is_Ok()
      ==> (old_walk.rtte.ripas == EMPTY ||
           old_walk.rtte.ripas == RAM));
  // Dependency on HIPAS (HIPAS is ASSIGNED)
  assert(res.is_Ok()
      ==> old_walk.rtte.state == ASSIGNED);
  // New RIPAS (If RIPAS is EMPTY)
  assert(res.is_Ok() && old_walk.rtte.ripas == EMPTY
      ==> old_walk.rtte.ripas == new_walk.rtte.ripas);
  // New RIPAS (If RIPAS is RAM)
  assert(res.is_Ok() && old_walk.rtte.ripas == RAM
      ==> new_walk.rtte.ripas == DESTROYED);
  // New HIPAS
  assert(res.is_Ok()
      ==> new_walk.rtte.state == UNASSIGNED);
}
```

Summary conversion

RMI command	Dependency on RIPAS	Dependency on HIPAS	New RIPAS	New HIPAS
DATA_DESTROY	EMPTY	ASSIGNED	Unchanged	UNASSIGNED
DATA_DESTROY	RAM	ASSIGNED	DESTROYED	UNASSIGNED

Dependency of RMI command execution,
From the RMM specification (1.0-eac5).

```
pub proof fn rmi_data_destroy_rule (rd: Address,
  ipa: Address, res: Result<(), RmiStatusCode>,
  data: Address, top: Address, old_s: S, new_s: S)
    requires rmi_data_destroy_spec(rd, ipa, res,
                                   data, top, old_s, new_s),
{
  let old_walk = RttWalk_(old_s, rd, ipa,
                          RMM_RTT_PAGE_LEVEL);
  let new_walk = RttWalk_(new_s, rd, ipa,
                          RMM_RTT_PAGE_LEVEL);
  // Dependency on RIPAS (If RIPAS is EMPTY or RAM)
  // The below would trigger an assertion violation
  assert(res.is_Ok()
      ==> (old_walk.rtte.ripas == EMPTY ||
           old_walk.rtte.ripas == RAM));
  // Dependency on HIPAS (HIPAS is ASSIGNED)
  assert(res.is_Ok()
      ==> old_walk.rtte.state == ASSIGNED);
  // New RIPAS (If RIPAS is EMPTY)
  assert(res.is_Ok() && old_walk.rtte.ripas == EMPTY
      ==> old_walk.rtte.ripas == new_walk.rtte.ripas);
  // New RIPAS (If RIPAS is RAM)
  assert(res.is_Ok() && old_walk.rtte.ripas == RAM
      ==> new_walk.rtte.ripas == DESTROYED);
  // New HIPAS
  assert(res.is_Ok()
      ==> new_walk.rtte.state == UNASSIGNED);
}
```

Summary conversion

RMI command	Dependency on RIPAS	Dependency on HIPAS	New RIPAS	New HIPAS
DATA_DESTROY	EMPTY	ASSIGNED	Unchanged	UNASSIGNED
DATA_DESTROY	RAM	ASSIGNED	DESTROYED	UNASSIGNED

Dependency of RMI command execution,
From the RMM specification (1.0-eac5).

```
pub proof fn rmi_data_destroy_rule (rd: Address,
  ipa: Address, res: Result<(), RmiStatusCode>,
  data: Address, top: Address, old_s: S, new_s: S)
    requires rmi_data_destroy_spec(rd, ipa, res,
                                   data, top, old_s, new_s),
{
  let old_walk = RttWalk_(old_s, rd, ipa,
                          RMM_RTT_PAGE_LEVEL);
  let new_walk = RttWalk_(new_s, rd, ipa,
                          RMM_RTT_PAGE_LEVEL);
  // Dependency on RIPAS (If RIPAS is EMPTY or RAM)
  // The below would trigger an assertion violation
  assert(res.is_Ok()
      ==> (old_walk.rtte.ripas == EMPTY ||
           old_walk.rtte.ripas == RAM));
  // Dependency on HIPAS (HIPAS is ASSIGNED)
  assert(res.is_Ok()
      ==> old_walk.rtte.state == ASSIGNED);
  // New RIPAS (If RIPAS is EMPTY)
  assert(res.is_Ok() && old_walk.rtte.ripas == EMPTY
      ==> old_walk.rtte.ripas == new_walk.rtte.ripas);
  // New RIPAS (If RIPAS is RAM)
  assert(res.is_Ok() && old_walk.rtte.ripas == RAM
      ==> new_walk.rtte.ripas == DESTROYED);
  // New HIPAS
  assert(res.is_Ok()
      ==> new_walk.rtte.state == UNASSIGNED);
}
```

Rule-based checks

- Heuristic rules inspect interrelated components for bug detection.

- Footprint checks: If a state modification occurs in success conditions but is not reflected in the footprint, the discrepancy likely indicates an inconsistency.

- Dangling output checks: If an output value is absent from all postconditions, we flag it as a potential inconsistency.

Implementation

- Scope is implemented in python script (2k lines).
  - Uses pdftotext tool for the initial PDF parsing.

- The script uses a top-down approach for parsing document structures (part/chapter/section).

- For additional parsing (e.g., multi-line table entries), regular expressions and heuristics are used.

Evaluation

1. How effectively does Scope detect inconsistencies in the RMM specification?

2. How does Scope perform relative to existing approaches?

3. How robust is Scope across different versions of the specification?

4. What security risks arise from inconsistencies?

Evaluation (Effectiveness)

- Discovered 35 bugs, all confirmed by Arm.
  - Formal reasoning: 10 cases
  - Rule-based checking: 25 cases (footprint: 13, dangling: 12)
  - Bugs in recent commands: 13 cases

- Achieved the precision of 63.64% and 61.90% for 1.0-eac5 and 1.0-rel0.
  - For formal reasoning, 33.33% and 25% were measured.
  - For rule-based checking, 84.62% was measured for both versions.

Evaluation (LLM comparison)

> "You are an expert in ARM CCA and PDF document. (…) Find all the inconsistencies inside the input text, with detailed explanations."

- Used chat models (GPT-4o, Claude3.7, Llama3.1-70B) and reasoning models (GPT-o1, Deepseek-R1).

- Developed an automatic context-splitting agent that partitions the document based on its internal hyperlinks and structure.

Evaluation (LLM comparison)

Models		Result*	Precision
Chat Model	Llama3.1	1/66	1.52%
	Gpt-4o	1/54	1.85%
	Claude-3.7	2/57	3.51%
Reasoning Model	Gpt-o1	4/50	8.00%
	Deepseek-r1	1/53	1.89%
SCOPE		13/21	61.90%

Evaluation between SCOPE and LLM based approaches.
SCOPE effectively improves the precision by 7-40x compared to the LLM-based approaches. *: Result = TP/(TP+FP)

Evaluation (Robustness)

- Measured covered rates on multiple versions of the RMM specification.

Spec Version	1.0-eac5	1.0-rel0	1.1-alp11	1.1-alp12
VIA (OSDI22)	22 (54%)	N/A	N/A	N/A
Arm's model checking	8 (20%)	8 (20%)	N/A	N/A
SCOPE	28 (68%)	28 (68%)	74 (77%)	79 (78%)
Total	41	41	96	101

Number of successfully covered ABI commands.
N/A indicates no covered rate due to the absence of an implementation in that version.

Evaluation (Security implications)

Before

After

B4.3.16.3 Success conditions

ID	Condition
rtte_state	`walk.rtte.state == UNASSIGNED`
ripas	`walk.rtte.ripas == DESTROYED`
state_prot	`pre: AddrIsProtected(ipa, realm)` `post: walk.rtte.state == UNASSIGNED`
ripas	`pre: AddrIsProtected(ipa, realm)` `post: walk.rtte.ripas == DESTROYED`
state_unprot	`pre: !AddrIsProtected(ipa, realm)` `post: walk.rtte.state == UNASSIGNED_NS`

Buggy success conditions of RMI_RTT_DESTROY excerpted from the RMM specification (1.0-eac5). The invariants for Protected IPAs and Unprotected IPAs are not considered.

Evaluation (Security implications)

Name	Description
UNASSIGEND	This RTTE is identified by a Protected IPA.
UNASSIGNED_NS	This RTTE is identified by a Unprotected IPA.

B4.3.16.3 Success conditions

ID	Condition
rtte_state	`walk.rtte.state == UNASSIGNED`
ripas	`walk.rtte.ripas == DESTROYED`
state_prot	`pre: AddrIsProtected(ipa, realm)` `post: walk.rtte.state == UNASSIGNED`
ripas	`pre: AddrIsProtected(ipa, realm)` `post: walk.rtte.ripas == DESTROYED`
state_unprot	`pre: !AddrIsProtected(ipa, realm)` `post: walk.rtte.state == UNASSIGNED_NS`

Before

After

Buggy success conditions of RMI_RTT_DESTROY excerpted from the RMM specification (1.0-eac5). The invariants for Protected IPAs and Unprotected IPAs are not considered.

Evaluation (Security implications)

Before

After

B4.3.16.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED
ripas	walk.rtte.ripas == DESTROYED
state_prot	pre: AddrIsProtected(ipa, realm) post: walk.rtte.state == UNASSIGNED
ripas	pre: AddrIsProtected(ipa, realm) post: walk.rtte.ripas == DESTROYED
state_unprot	pre: !AddrIsProtected(ipa, realm) post: walk.rtte.state == UNASSIGNED_NS

The previous condition could cause a risk of unauthorized access to protected memory regions.

Buggy success conditions of RMI_RTT_DESTROY excerpted from the RMM specification (1.0-eac5). The invariants for Protected IPAs and Unprotected IPAs are not considered.

Conclusion

- An automated framework detecting inconsistencies in the RMM specification, uncovering 35 previously unknown bugs.
- New methodology: operating solely on specification documents, while handling frequent updates.
- The source code for Scope is publicly available (github.com/islet-project/scope).

Q&A