Systematic API Testing through Model Checking and Executable Contracts

TLA+ Community Meeting

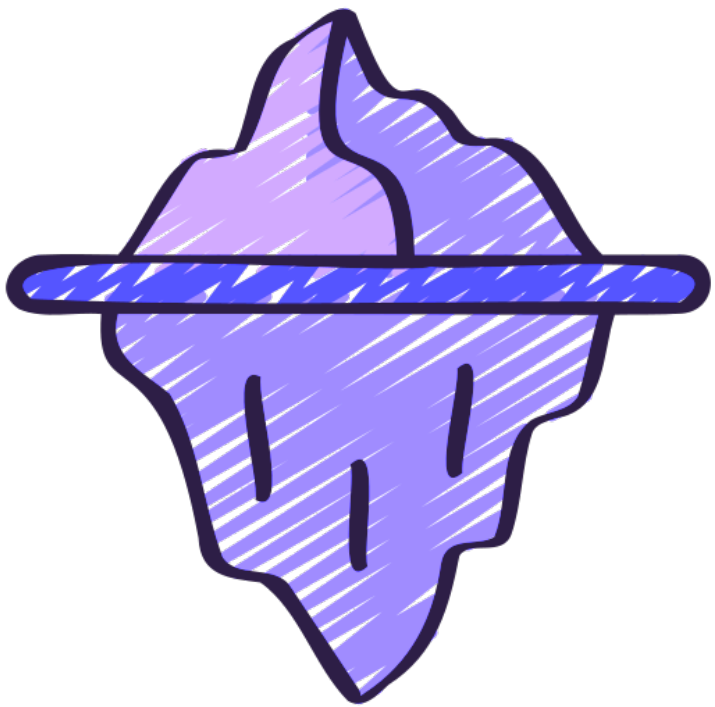
Ana Catarina Ribeiro

NOVA University Lisbon

April 12, 2026

Cracking the Ice:

How to test an iceberg?



The Iceberg *aka “The SUT”*

Implementation

Heterogeneous

Black-box

The Iceberg *aka “The SUT”*

API

RESTful

Implementation

Heterogeneous

Black-box

The Iceberg *aka "The SUT"*

API

RESTful

REpresentational State Transfer (REST)

Guidelines on building web services on top of HTTP.

The Iceberg *aka "The SUT"*

API

RESTful

Described in OAS

The Iceberg *aka "The SUT"*

API

RESTful

Described in OAS

OAS (Open API Specification)

Standard, language-agnostic format for specifying HTTP APIs.

The Iceberg *aka "The SUT"*

API

RESTful

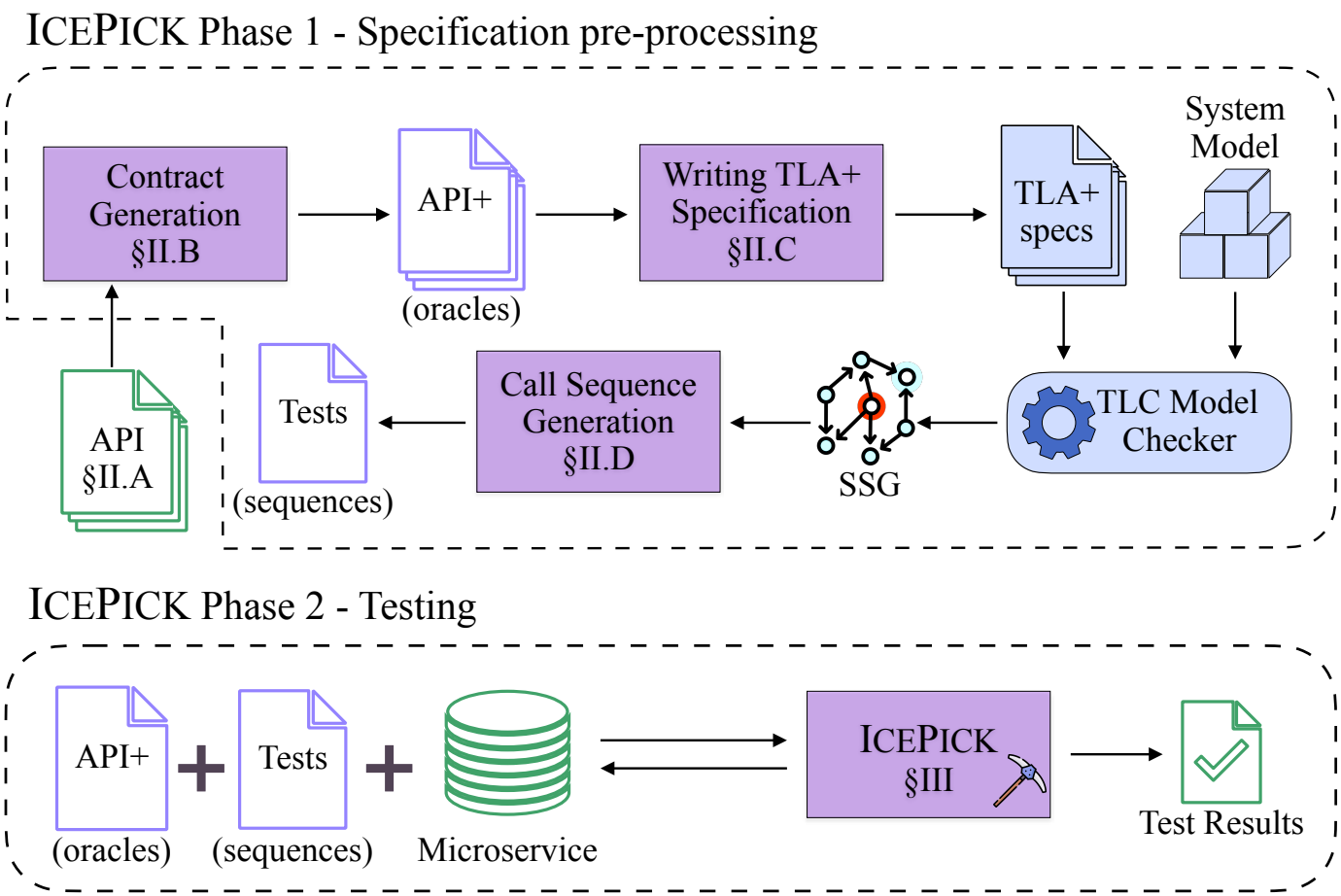
Described in OAS

Implementation

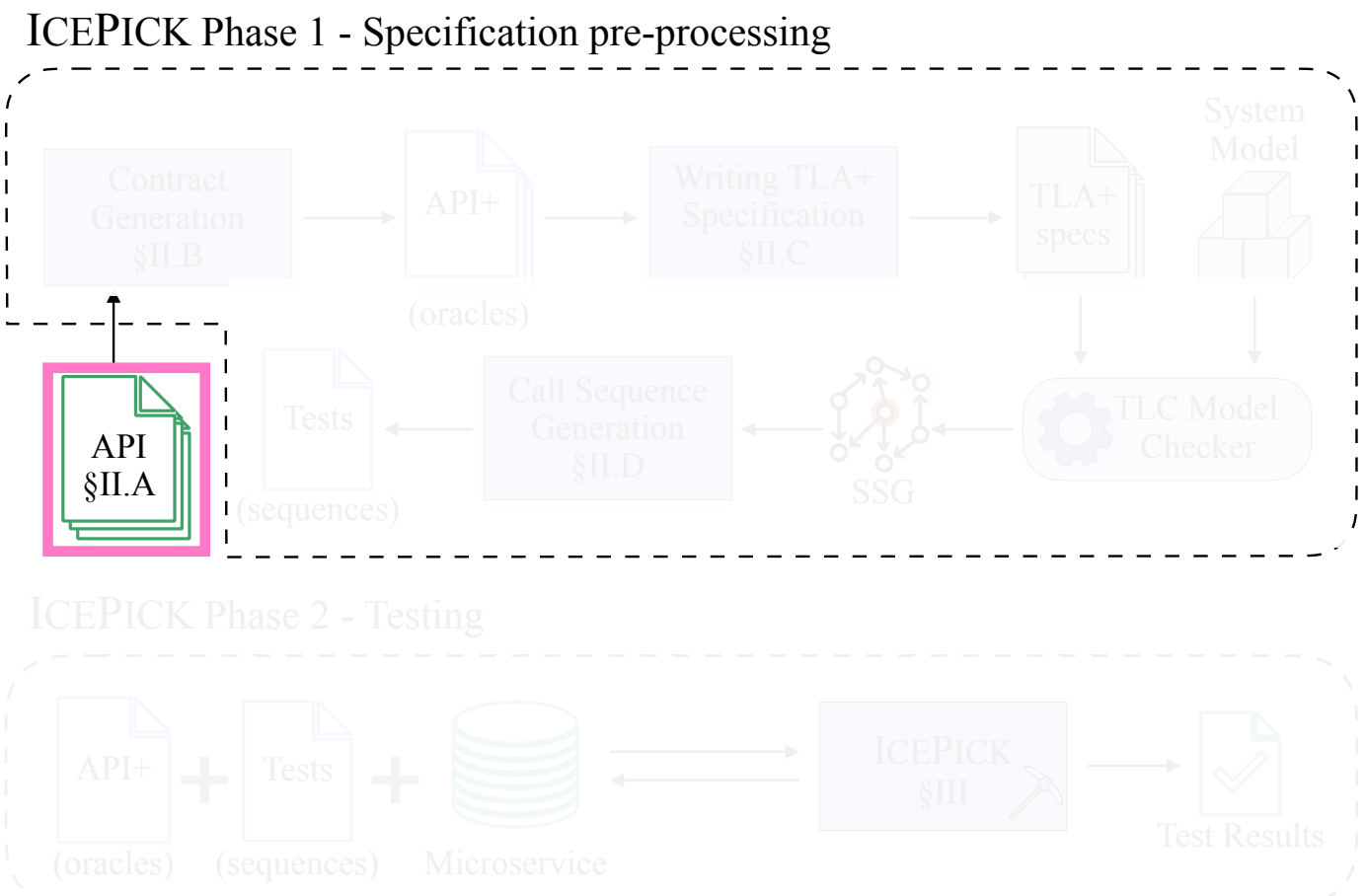
Heterogeneous

Black-box

Solution – Overview



Phase 1 – The Standard



OpenAPI Specification

OAS offers:

- operation endpoints
- expected data types
- scenario-based response codes

OpenAPI Specification

OAS offers:

- operation endpoints
- expected data types
- scenario-based response codes

Running Example

`Tournament`
- `tid`
- `capacity (c)`
- `players (ps)`

`Player`
- `pid`
- `name`
- `tournaments (ts)`

`Enrolments`
- `eid`
- `pid`
- `tid`

OpenAPI Specification

OAS offers:

- operation endpoints
- expected data types
- scenario-based response codes

```yaml
/players:
  post:
    summary: Add a new player.
    operationID: postPlayer
    requestBody:
      content:
      application/json:
      schema:
        $ref: #/schemas/Player
        required: true
    responses:
      200:
        description: Successful insertion.
        content:
          application/json:
            schema:
              $ref: #/schemas/Player
```

OpenAPI Specification – Not enough?

OAS offers:

- operation endpoints
- expected data types
- scenario-based response codes

Existing solutions derive tests exclusively from OAS specifications.

- poorly written (param. naming)
- incomplete (op. dependencies)
- outdated (evolution)

```
/players:
  post:
    summary: Add a new player.
    operationID: postPlayer
    requestBody:
      content:
      application/json:
      schema:
        $ref: #/schemas/Player
        required: true
    responses:
      200:
        description: Successful insertion.
        content:
          application/json:
            schema:
              $ref: #/schemas/Player
```

OpenAPI Specification – Not enough?

OAS offers:

- operation endpoints
- expected data types
- scenario-based response codes

Existing solutions derive tests exclusively from OAS specifications.

- poorly written (param. naming)
- incomplete (op. dependencies)
- outdated (evolution)

- status-code oracles (**5xx ISE?**)

```yaml
responses:
  500:
    description: Unexpected error.
    content:
      application/json:
        schema:
        $ref: "#/schemas/ErrorObject"
```

OpenAPI Specification – Not enough?

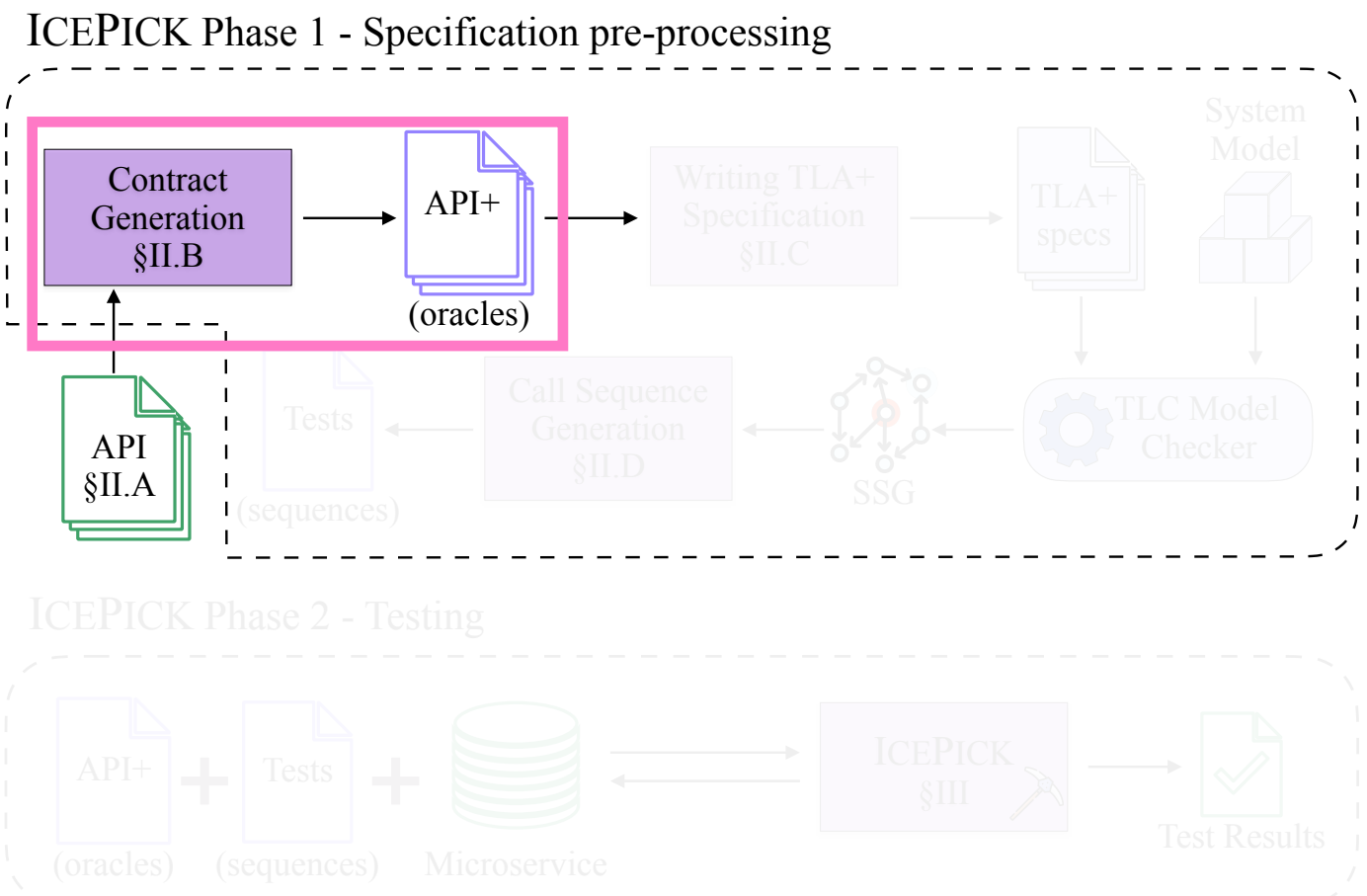
OAS offers:

- operation endpoints
- expected data types
- scenario-based response codes

OAS lacks detail to capture system behaviour

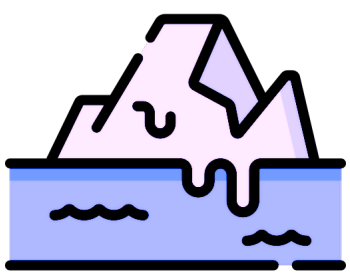
- preconditions
- postconditions
- invariants

Phase 1 – Extending the Standard



GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

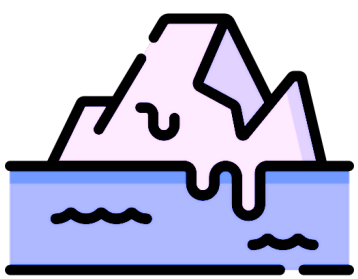


Enriches API specifications with practical properties for test generation

- Aims to solve the oracle problem, by adding
  - preconditions
  - postconditions
  - invariants
- RESTful API tailored constructs:
  - `res_code`, `res_body` / `req_body`
- Useful spec constructs
  - `@`, `prev`

GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

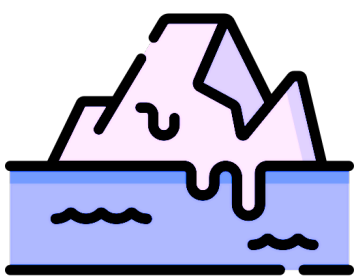


Enriches API specifications with practical properties for test generation

```
/players:
 POST:
  operationID: postPlayer
  requires:
  - res_code(GET /players/req_body(@){pid}) = 404
  ensures:
  - res_code(GET /players/req_body(@){pid}) = 200
  - req_body(@) = res_body(@)
```

GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

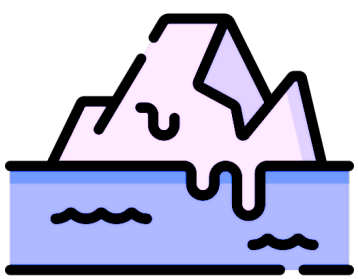


Enriches API specifications with practical properties for test generation

```
/players:
 POST:
  operationID: postPlayer
  requires:
  - res_code(GET /players/req_body(@){pid}) = 404
  ensures:
  - res_code(GET /players/req_body(@){pid}) = 200
  - req_body(@) = res_body(@)
```

GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

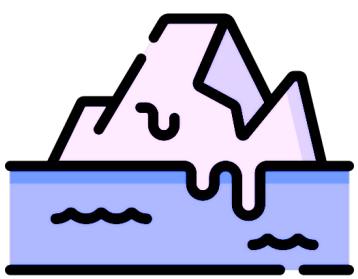


Enriches API specifications with practical properties for test generation

```
/players:
 POST:
  operationID: postPlayer
  requires:
  - res_code(GET /players/req_body(@){pid}) = 404
  ensures:
  - res_code(GET /players/req_body(@){pid}) = 200
  - req_body(@) = res_body(@)
```

GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

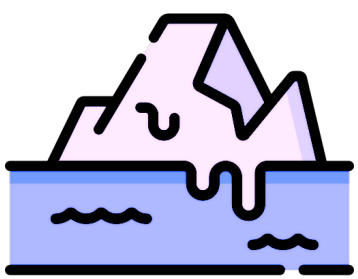


Enriches API specifications with practical properties for test generation

```
/players:
 POST:
  operationID: postPlayer
  requires:
  - res_code(GET /players/req_body(@){pid}) = 404
  ensures:
  - res_code(GET /players/req_body(@){pid}) = 200
  - req_body(@) = res_body(@)
```

GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

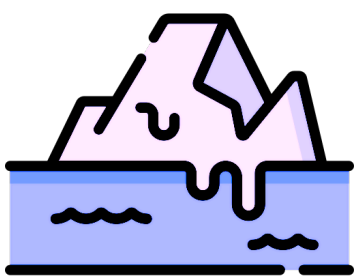


Enriches API specifications with practical properties for test generation

```
/players/{pid}:
 DELETE:
  operationID: deletePlayer
  requires:
  - res_code(GET /players/{pid}) = 200
  ensures:
  - res_code(GET /players/{pid}) = 404
  - req_body(@) = prev(res_body(GET /players/{pid}))
```

GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

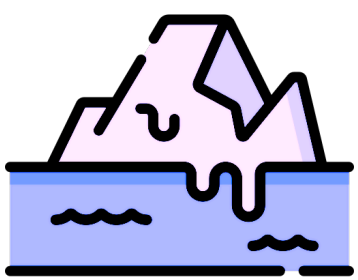


Enriches API specifications with practical properties for test generation

```
/players/{pid}:
 DELETE:
  operationID: deletePlayer
  requires:
  - res_code(GET /players/{pid}) = 200
  ensures:
  - res_code(GET /players/{pid}) = 404
  - req_body(@) = prev(res_body(GET /players/{pid}))
```

GLACIER – Yet another specification language?

First-order logic based contract specification language for RESTful APIs

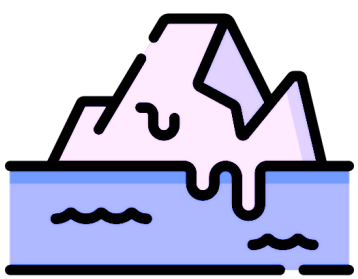


Enriches API specifications with practical properties for test generation

Invariants?

GLACIER – Yet another specification language?

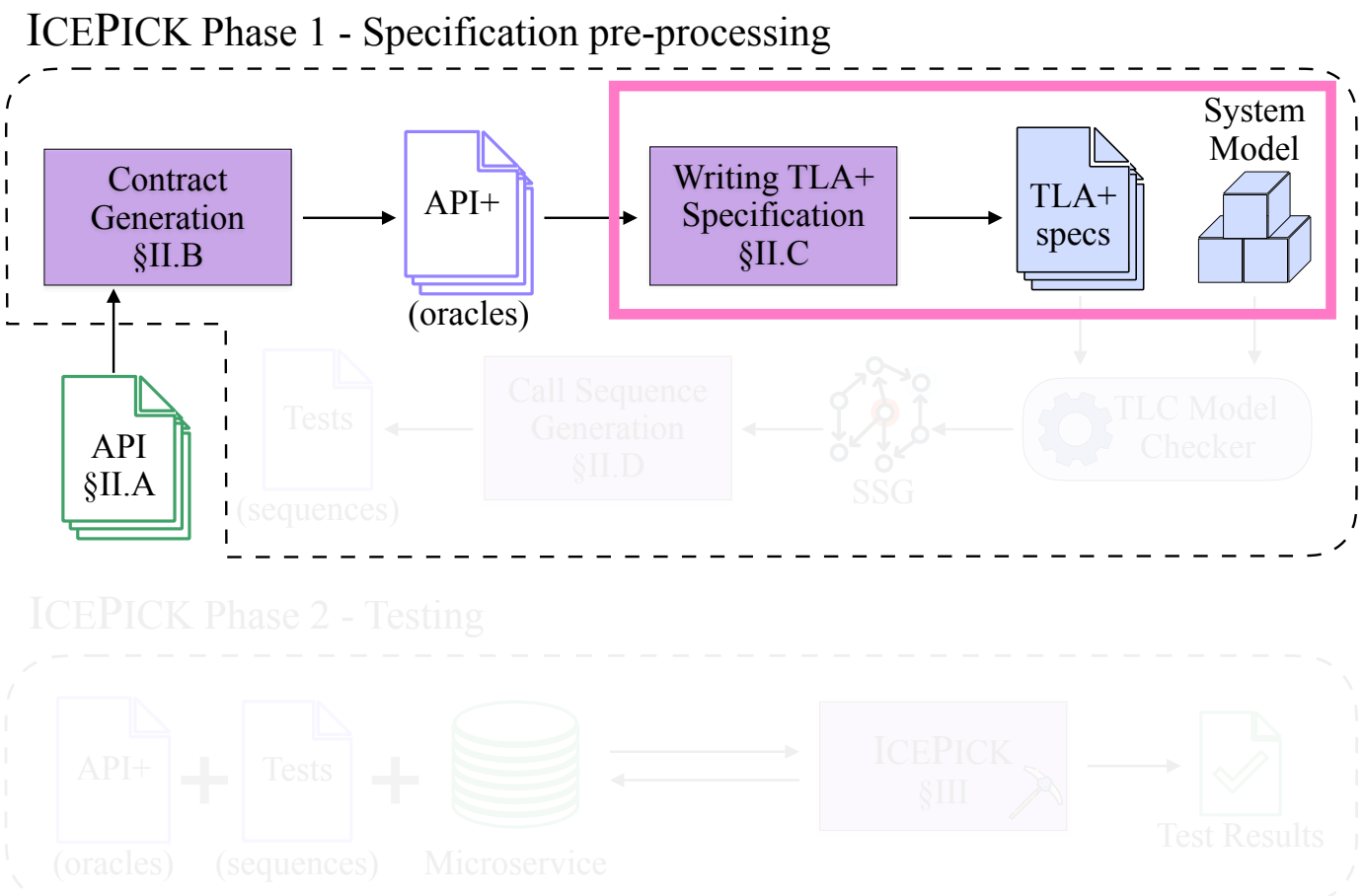
First-order logic based contract specification language for RESTful APIs



Enriches API specifications with practical properties for test generation

```
invariants:
- for t in resp_body(GET /tournaments) :-
  res_body(GET /tournaments/{t.tid}/players).len ⩽
  res_body(GET /tournaments/{t.tid}/capacity)
```

Phase 1 – The TLA+ Specification



Phase 1: TLA+ Specification – The Abstraction

- Life-cycle oriented
  - `create` → `update` → `remove`

Phase 1: TLA+ Specification – The Abstraction

- Life-cycle oriented
  - `create` → `update` → `remove`
- Focus on operations inducing state transitions
  - `post`
  - `delete`

Phase 1: TLA+ Specification – The Abstraction

- Life-cycle oriented
  - `create` → `update` → `remove`
- Focus on operations inducing state transitions
  - `post`
  - `delete`
- Read-only operations are not considered
  - `get`

Phase 1: TLA+ Specification – The Abstraction

- Life-cycle oriented
  - `create` → `update` → `remove`
- Focus on operations inducing state transitions
  - `post`
  - `delete`
- Read-only operations are not considered
  - `get`
- Update operations modifying abstracted fields are handled at a later stage
  - `put`
  - `patch`

Phase 1: Tournaments API – TLA+ Specification

```
CONSTANTS PID, TID, EID, N
```

Resource identifiers and the Naturals set.

Phase 1: Tournaments API – TLA+ Specification

```
CONSTANTS PID, TID, EID, N
```

Resource identifiers and the Naturals set.

```
ASSUME
  ∧ PID ≠ {}
  ∧ TID ≠ {}
  ∧ EID ≠ {}
  ∧ N ∈ SUBSET (Nat \ {0})
  ∧ IsFiniteSet(N)
```

All identifier sets are non-empty and N is a finite set of positive natural numbers.

Phase 1: Tournaments API – TLA+ Specification

```
PlayerType     == [ts: SUBSET TID]
TournamentType == [ps: SUBSET PID, c: N]
EnrolmentType  == [pid: PID, tid: TID]
```

Resources are represented as records, omitting schema fields irrelevant to reachability.

Phase 1: Tournaments API – TLA+ Specification

```
PlayerType     == [ts: SUBSET TID]
TournamentType == [ps: SUBSET PID, c: N]
EnrolmentType  == [pid: PID, tid: TID]
```

Resources are represented as records, omitting schema fields irrelevant to reachability.

```
VARIABLES players, tournaments, enrolments, final
```

One variable per resource type and a terminal state flag.

Phase 1: Tournaments API – TLA+ Specification

```
TypeInv ==
  ∧ IsMap(PID, PlayerType, players)
  ∧ IsMap(TID, TournamentType, tournaments)
  ∧ IsMap(EID, EnrolmentType, enrolments)
  ∧ final ∈ BOOLEAN
```

Resource maps are keyed by their identifiers and `final` is a boolean.

Phase 1: Tournaments API – TLA+ Specification

```
TypeInv ==
  ∧ IsMap(PID, PlayerType, players)
  ∧ IsMap(TID, TournamentType, tournaments)
  ∧ IsMap(EID, EnrolmentType, enrolments)
  ∧ final ∈ BOOLEAN
```

Resource maps are keyed by their identifiers and `final` is a boolean.

```
Init ==
  ∧ players = EmptyMap
  ∧ tournaments = EmptyMap
  ∧ enrolments = EmptyMap
  ∧ final = FALSE
```

Initial state sets all resource maps to empty and `final` to `FALSE`.

Phase 1: Tournaments API – TLA+ Specification

```
Inv1 ==
  ∀ tid ∈ TID : tid ∈ DOMAIN tournaments ⇒
    Cardinality(tournaments[tid].ps) ≤ tournaments[tid].c
```

GLACIER invariants become TLC-checked state invariants.

Phase 1: Tournaments API – TLA+ Specification

```
Requires(pid) == pid ∉ DOMAIN players
Ensures(pid)  == pid ∈ DOMAIN players
```

Operational contracts become action guards and next-state constraints.

Phase 1: Tournaments API – TLA+ Specification

```
postPlayer(pid) ==
  ∧ Requires(pid)
  ∧ players' = MapPut(players, pid, [ts ↦ {}])
  ∧ Ensures(pid)
  ∧ final' = FinalState(players', tournaments, enrolments)
  ∧ UNCHANGED<<tournaments, enrolments>>
```

Each state-mutating API operation → TLA+ action.

Phase 1: Tournaments API – TLA+ Specification

```
postPlayer(pid) ==
  ∧ Requires(pid)
  ∧ players' = MapPut(players, pid, [ts ↦ {}])
  ∧ Ensures(pid)
  ∧ final' = FinalState(players', tournaments, enrolments)
  ∧ UNCHANGED<<tournaments, enrolments>>
```

Each state-mutating API operation → TLA+ action.

Phase 1: Tournaments API – TLA+ Specification

```
postPlayer(pid) ==
  ∧ Requires(pid)
  ∧ players' = MapPut(players, pid, [ts ↦ {}])
  ∧ Ensures(pid)
  ∧ final' = FinalState(players', tournaments, enrolments)
  ∧ UNCHANGED<<tournaments, enrolments>>
```

Each state-mutating API operation → TLA+ action.

Phase 1: Tournaments API – TLA+ Specification

```
postPlayer(pid) ==
  ∧ Requires(pid)
  ∧ players' = MapPut(players, pid, [ts ↦ {}])
  ∧ Ensures(pid)
  ∧ final' = FinalState(players', tournaments, enrolments)
  ∧ UNCHANGED<<tournaments, enrolments>>
```

Each state-mutating API operation → TLA+ action.

Phase 1: Tournaments API – TLA+ Specification

```
postPlayer(pid) ==
  ∧ Requires(pid)
  ∧ players' = MapPut(players, pid, [ts ↦ {}])
  ∧ Ensures(pid)
  ∧ final' = FinalState(players', tournaments, enrolments)
  ∧ UNCHANGED<<tournaments, enrolments>>
```

Each state-mutating API operation → TLA+ action.

Phase 1: Tournaments API – TLA+ Specification

```
postPlayer(pid) ==
  ∧ Requires(pid)
  ∧ players' = MapPut(players, pid, [ts ↦ {}])
  ∧ Ensures(pid)
  ∧ final' = FinalState(players', tournaments, enrolments)
  ∧ UNCHANGED<<tournaments, enrolments>>
```

Each state-mutating API operation → TLA+ action.

Phase 1: Tournaments API – TLA+ Specification

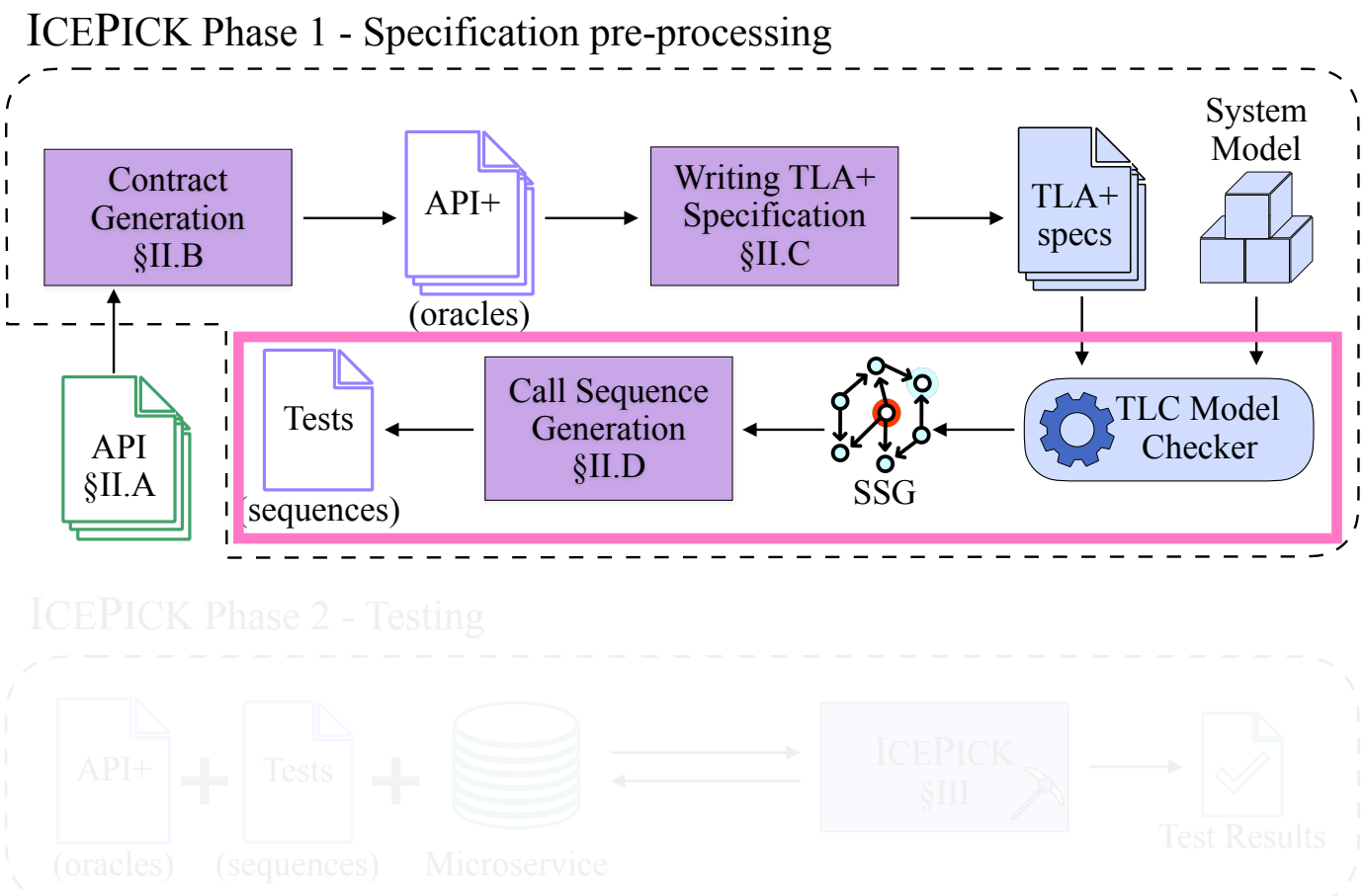
```
postPlayer(pid) ==
  ∧ Requires(pid)
  ∧ players' = MapPut(players, pid, [ts ↦ {}])
  ∧ Ensures(pid)
  ∧ final' = FinalState(players', tournaments, enrolments)
  ∧ UNCHANGED<<tournaments, enrolments>>
```

Each state-mutating API operation ⟶ TLA+ action.

```
FinalState(players, tournaments, enrolments) ==
  ∀ pid ∈ PID, tid ∈ TID, eid ∈ EID :
    ∧ pid ∈ DOMAIN players
    ∧ tid ∈ DOMAIN tournaments
    ∧ eid ∈ DOMAIN enrolments
```

We reach a final state when all resources (model values) have been explored.

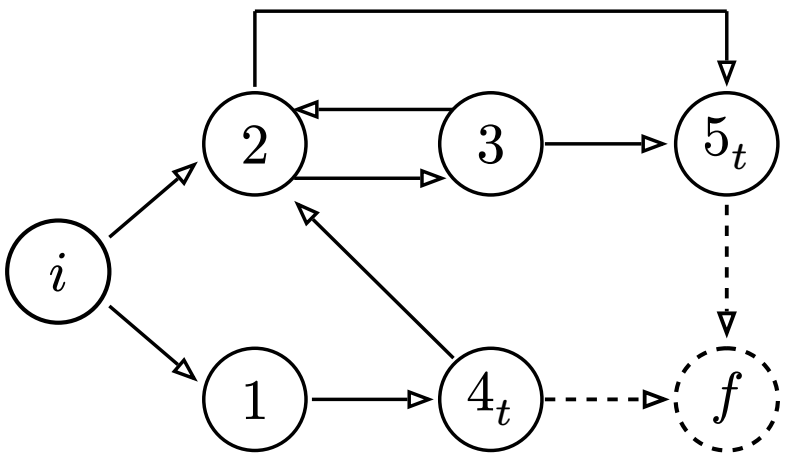
Phase 1 – Call Sequence Generation



Phase 1 – The State Space Graph (SSG)

The SSG is a directed graph $G = (V, E)$, where:

- V is the set of states explored by TLC
  - single initial state i
  - multiple terminal states
  - final sink state f
  - For every terminal state t, $(t, f) \in E$
- E is the set of state transitions (API operations)
  - The graph has no parallel edges



Phase 1 – Call Sequence Generation

Custom **coverage-guided** algorithm based on breadth-first search (BFS).

The algorithm works in three stages:

1. Collect paths from i, classifying them as complete (reaching f) or incomplete
2. Collect a path from each state to f
3. Complete the incomplete paths

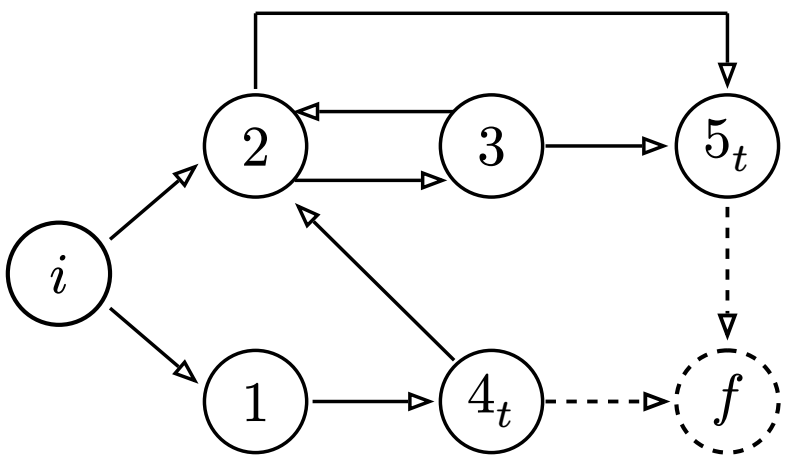
Phase 1 – Call Sequence Generation

Custom **coverage-guided** algorithm based on breadth-first search (BFS).

The algorithm works in three stages:

1. Collect paths from i, classifying them as complete (reaching f) or incomplete
2. Collect a path from each state to f
3. Complete the incomplete paths

Phase 1 – Call Sequence Generation (Stage 1)

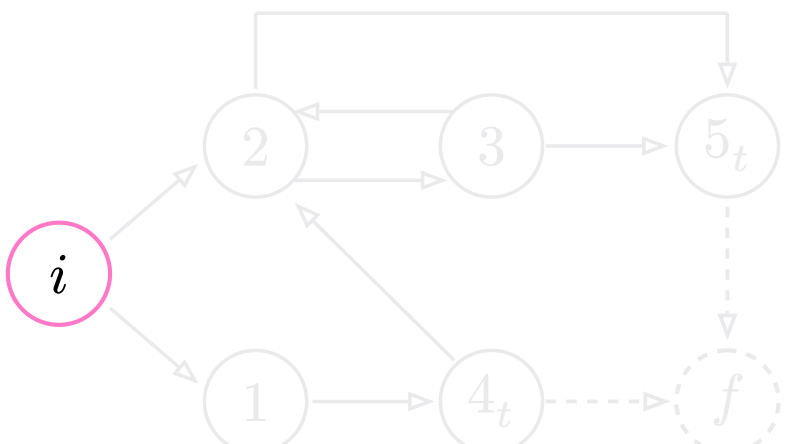


Phase 1 – Call Sequence Generation (Stage 1)

i $\langle i \rangle$

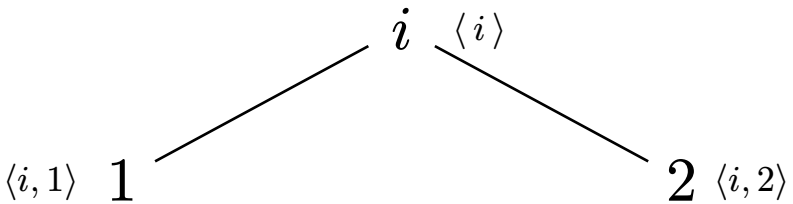
found: ⊤ | | | | | | ⊤

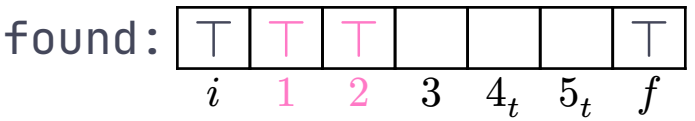
i 1 2 3 4_t 5_t f

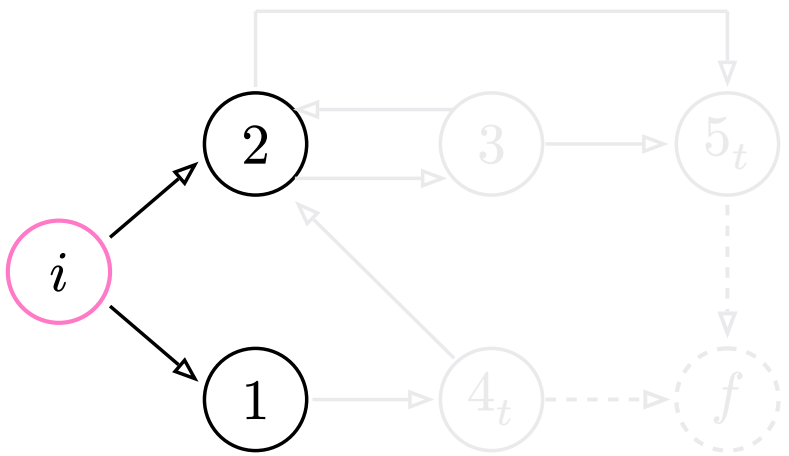


fifo: [i]
cmp : {}
inc : {}

Phase 1 – Call Sequence Generation (Stage 1)



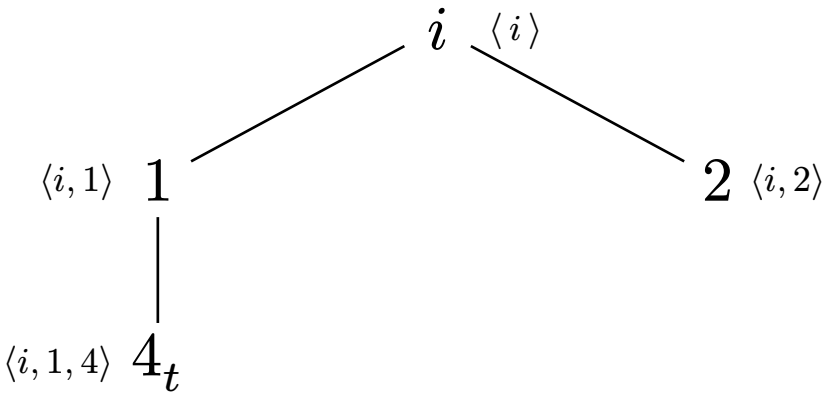




`fifo`: $[i, 1, 2]$
`cmp` : $\{\}$
`inc` : $\{\}$

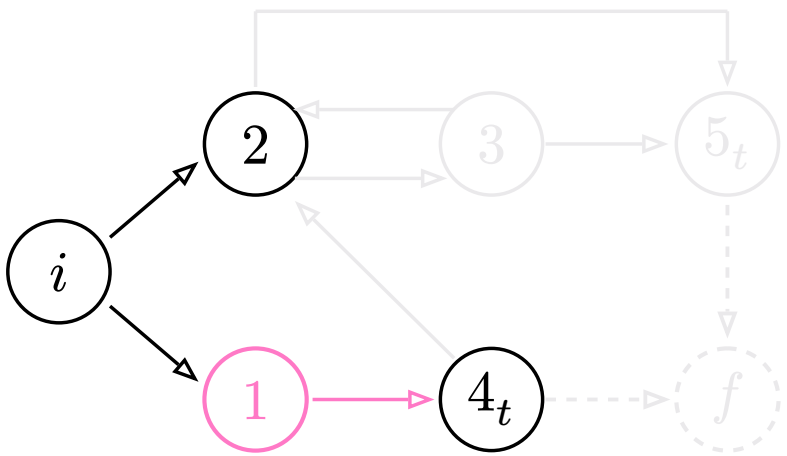
Expanding state i
We found two new states: 1 and 2
Mark as found; Add to `fifo`

Phase 1 – Call Sequence Generation (Stage 1)



found:

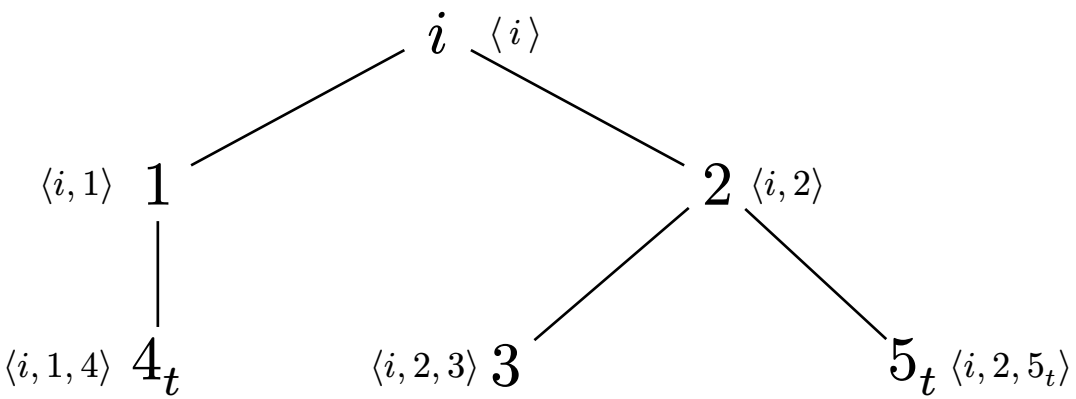
⊤	⊤	⊤		⊤		⊤
i	1	2	3	4_t	5_t	f

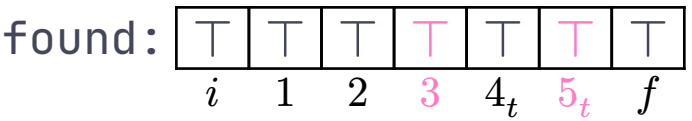


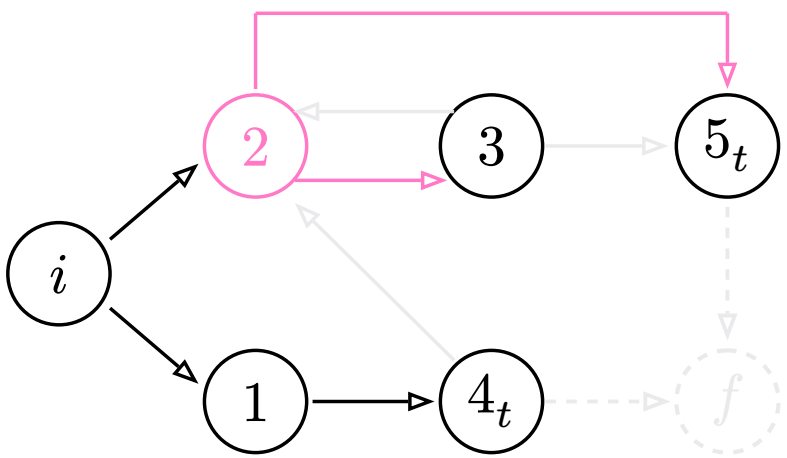
`fifo`: [i, 1, 2, 4_t]
`cmp` : {}
`inc` : {}

Expanding state 1
Found a new state: 4_t
Mark as found. Add to `fifo`

Phase 1 – Call Sequence Generation (Stage 1)



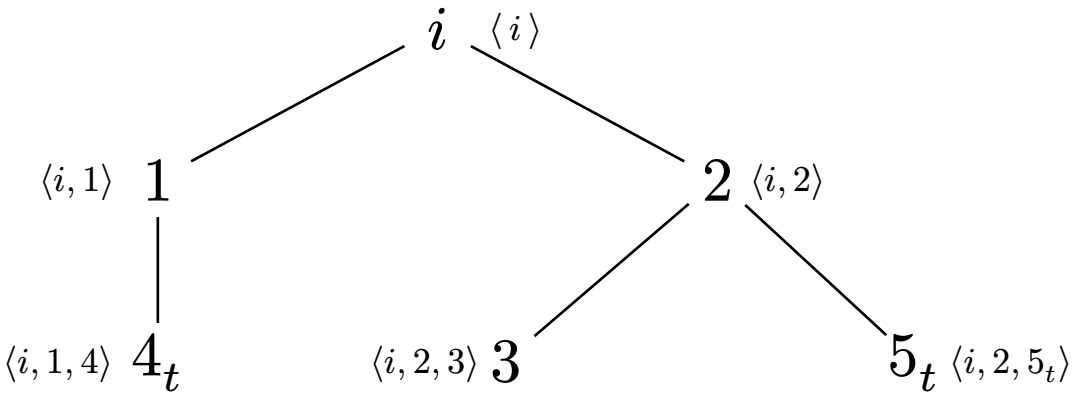


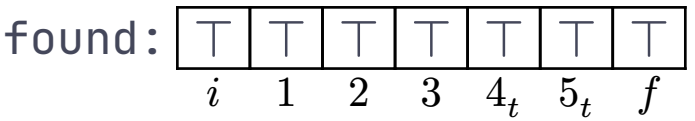


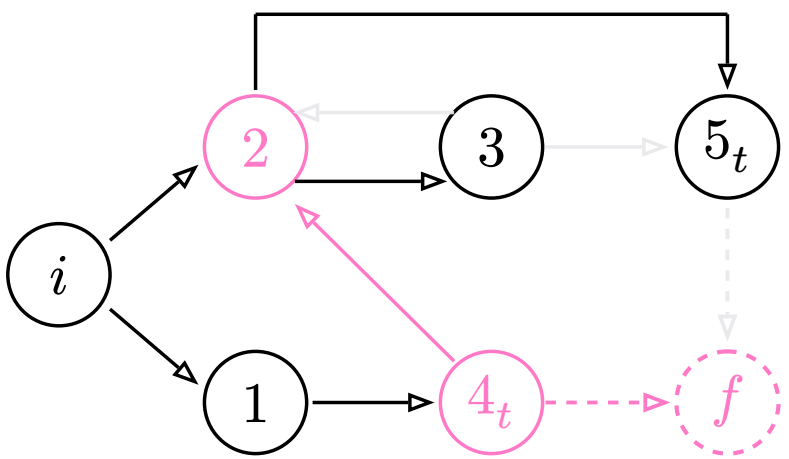
`fifo`: [i, 1, 2, 4_t, 3, 5_t]
`cmp` : {}
`inc` : {}

Expanding state 2
Found two new states: 3 and 5_t
Mark as found. Add to `fifo`

Phase 1 – Call Sequence Generation (Stage 1)



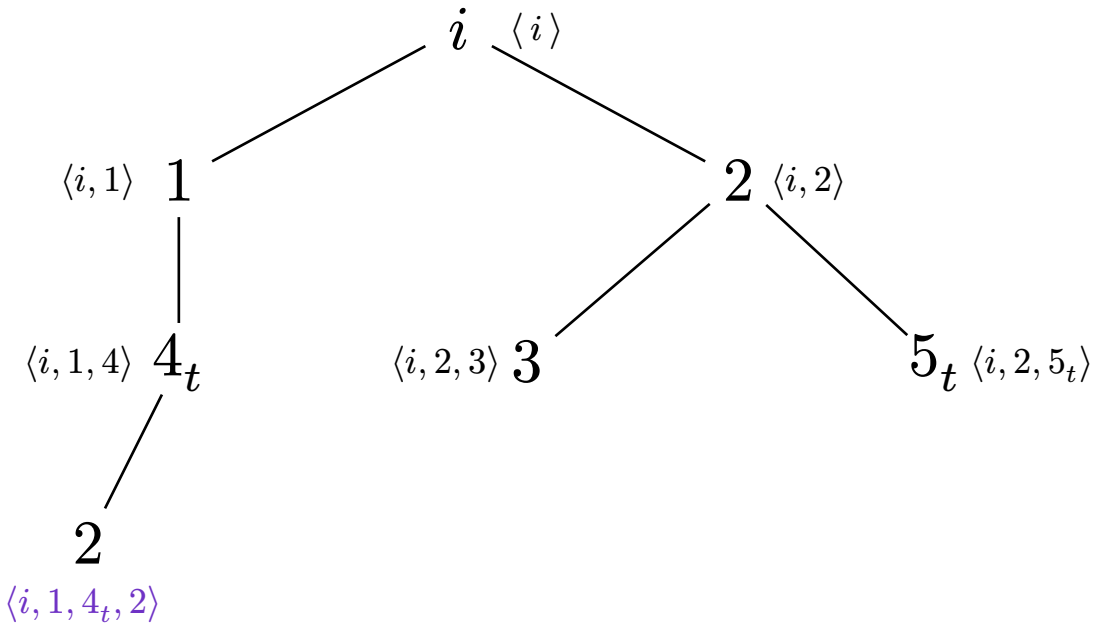


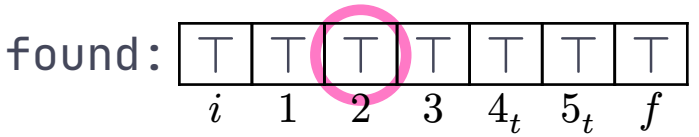


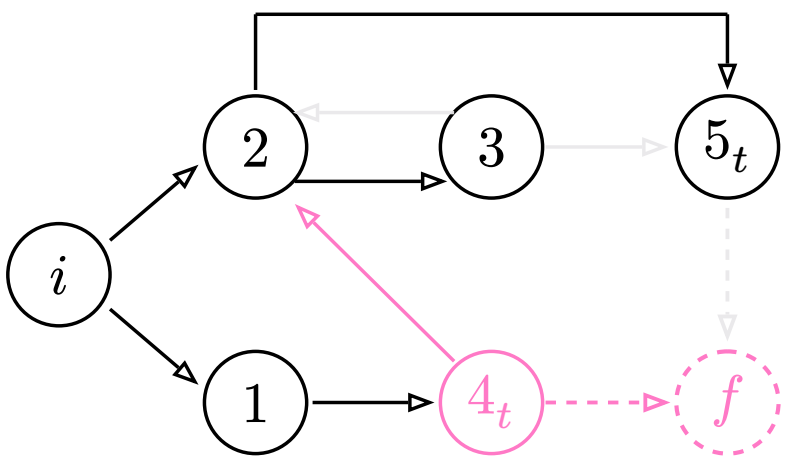
fifo: [i, 1, 2, 4_t, 3, 5_t]
cmp : {}
inc : {}

Expanding state 4_t

Phase 1 – Call Sequence Generation (Stage 1)



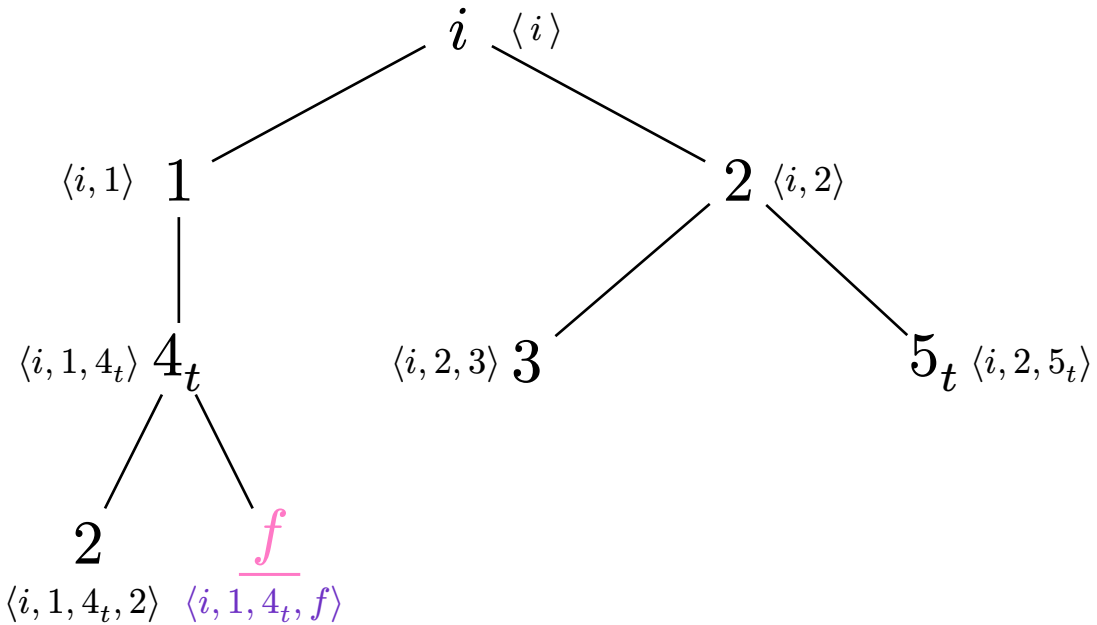


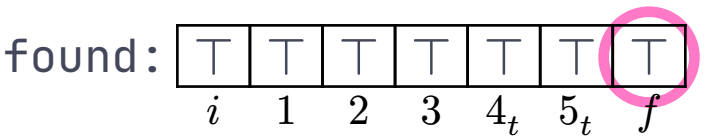


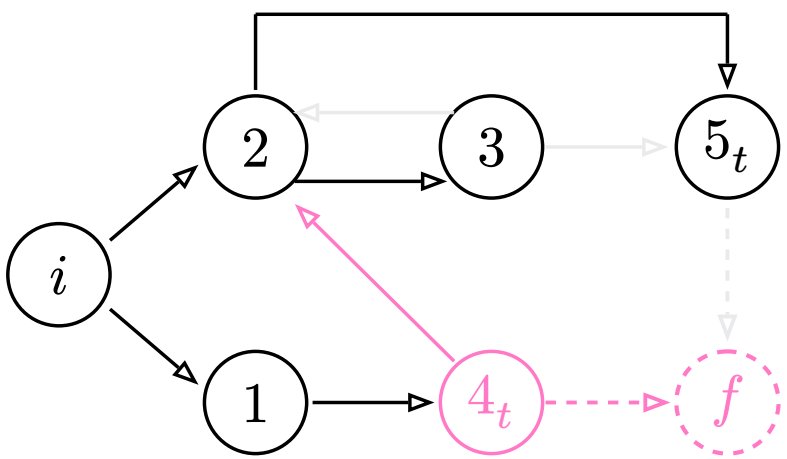
fifo: [i, 1, 2, 4_t, 3, 5_t]
cmp : {}
inc : { $\langle i, 1, 4_t, 2 \rangle$ }

Expanding state 4_t
State 2 has been found
Add path to **inc**

Phase 1 – Call Sequence Generation (Stage 1)



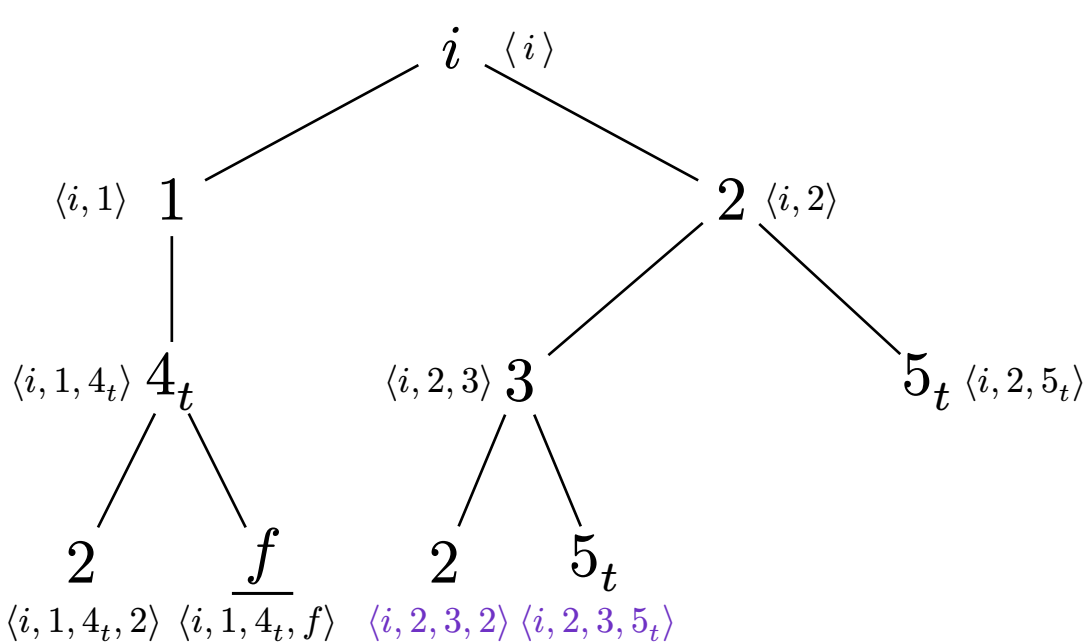


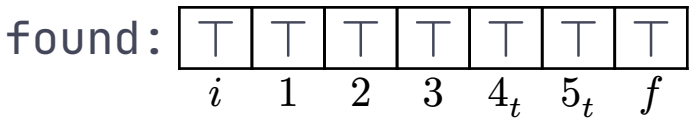


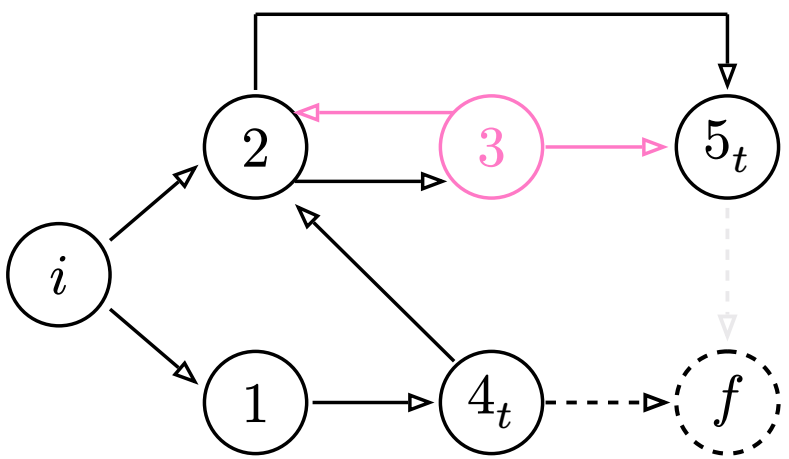
fifo: $[\, i, 1, 2, 4_t, 3, 5_t \,]$
cmp : $\{\, \langle i, 1, 4_t, f \rangle \,\}$
inc : $\{\langle i, 1, 4_t, 2 \rangle\}$

Expanding state 4_t
Final state f has been found
Add path to **cmp**

Phase 1 – Call Sequence Generation (Stage 1)



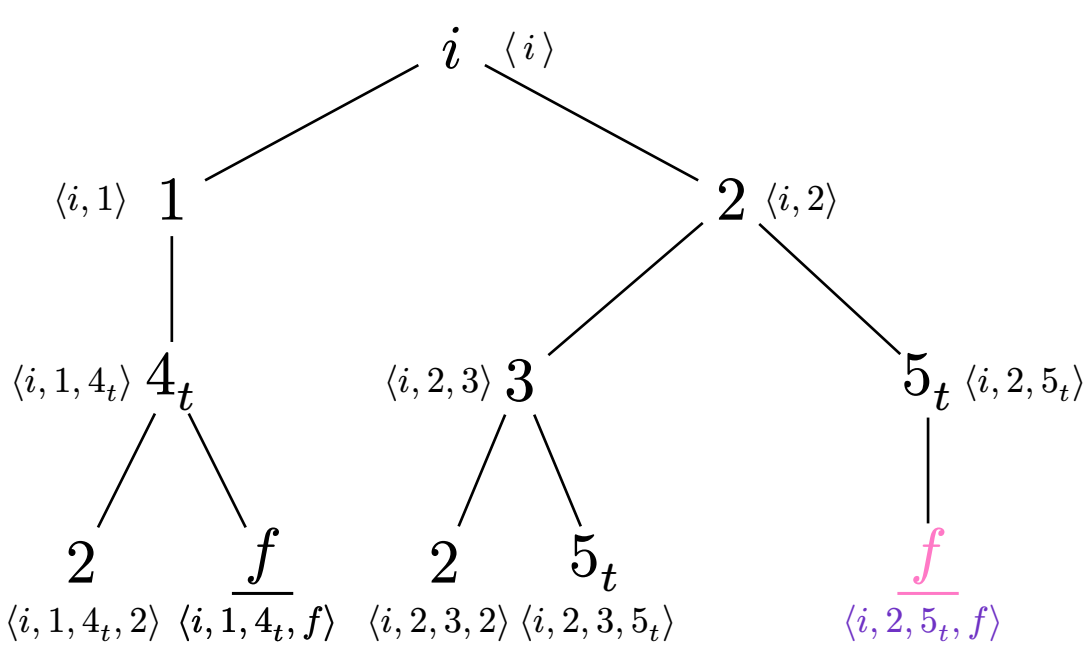


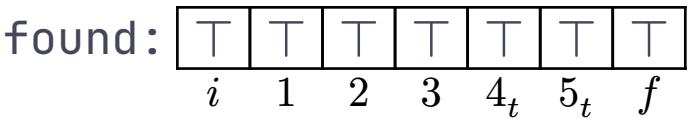


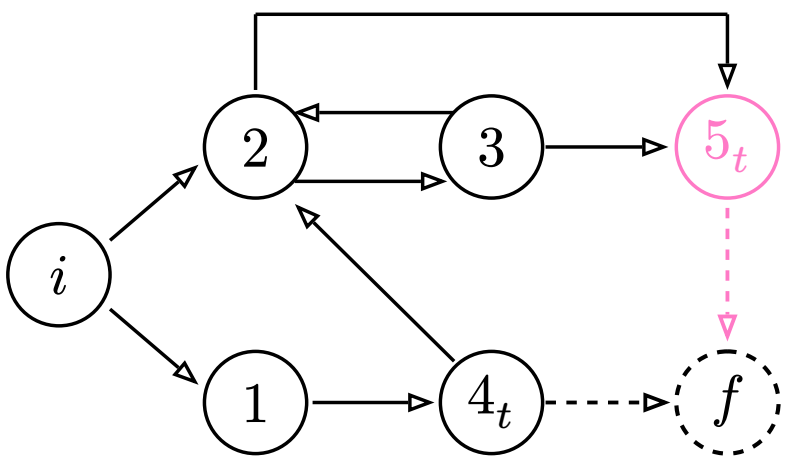
fifo: [$i, 1, 2, 4_t, 3, 5_t$]
cmp : $\{\langle i,1,4_t,f \rangle\}$
inc : $\{\langle i,1,4_t,2 \rangle, \langle i,2,3,2 \rangle, \langle i,2,3,5_t \rangle\}$

Expanding state 3
No new states
Add paths to **inc**

Phase 1 – Call Sequence Generation (Stage 1)



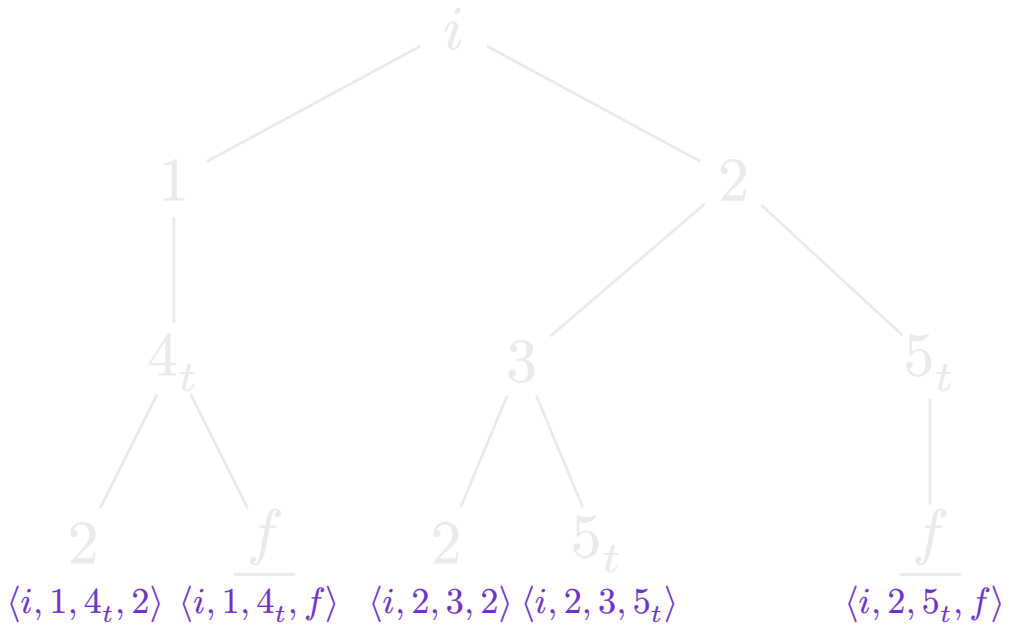




`fifo`: [$i, 1, 2, 4_t, 3, 5_t$]
`cmp` : $\{\langle i,1,4_t,f \rangle, \langle i,2,5_t,f \rangle\}$
`inc` : $\{\langle i,1,4_t,2 \rangle, \langle i,2,3,2 \rangle, \langle i,2,3,5_t \rangle\}$

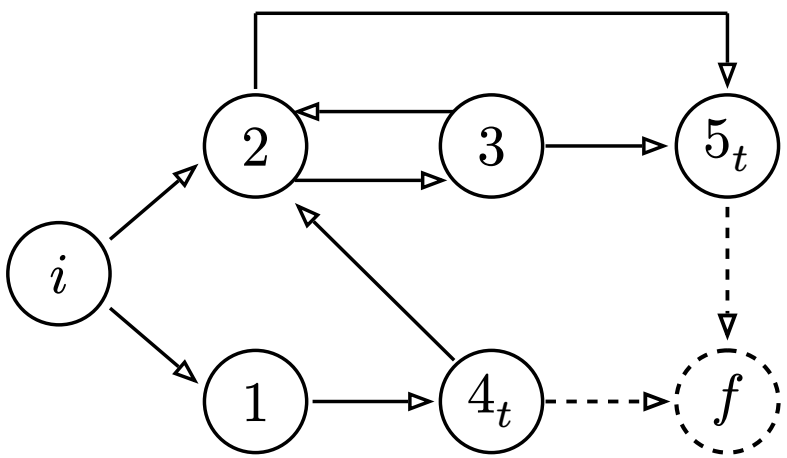
Expanding state 5_t
Final state f has been found
Add path to **`cmp`**

Phase 1 – Call Sequence Generation (Stage 1)



found: T T T T T T T

i 1 2 3 4_t 5_t f



fifo: $[\, i, 1, 2, 4_t, 3, 5_t \,]$

cmp : $\{\ \langle i, 1, 4_t, f\rangle, \langle i, 2, 5_t, f\rangle\ \}$

inc : $\{\ \langle i, 1, 4_t, 2\rangle, \langle i, 2, 3, 2\rangle, \langle i, 2, 3, 5_t\rangle\ \}$

By the end of stage 1 we know:

1. all states have been explored
2. all transitions belong to some path in **`cmp`** $\cup$ **`inc`**
3. we have the shortest paths from i to all nodes

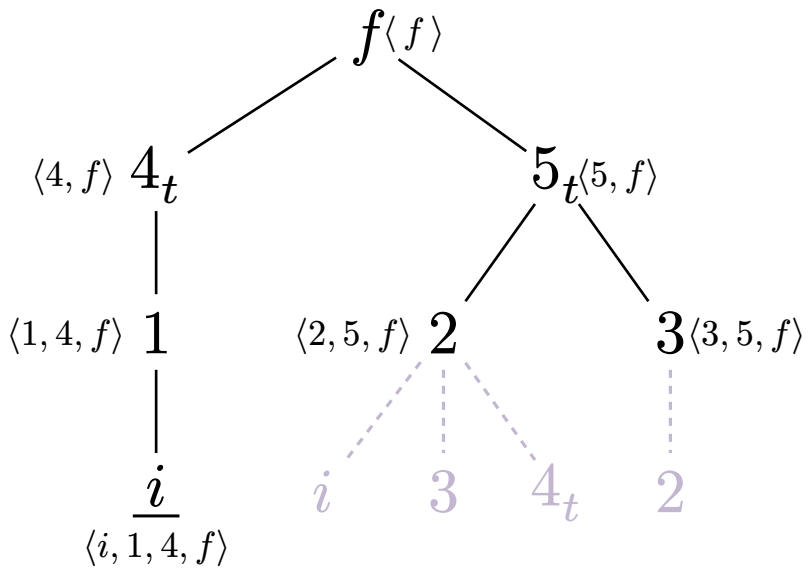
Phase 1 – Call Sequence Generation

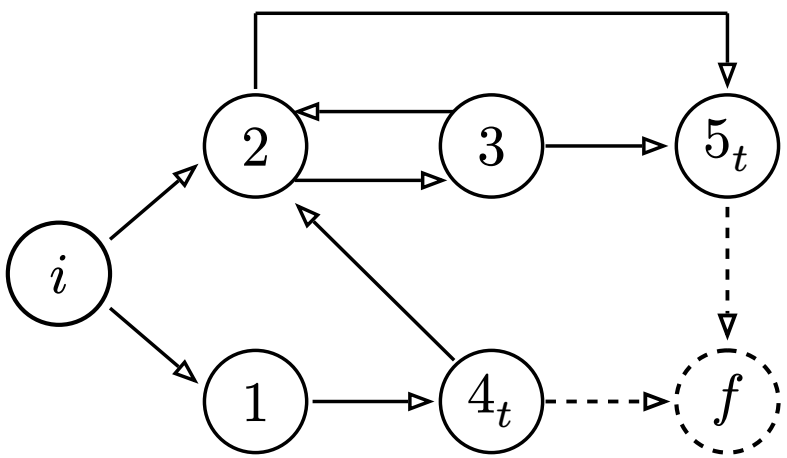
Custom **coverage-guided** algorithm based on breadth-first search (BFS).

The algorithm works in three stages:

1. Collect paths from i, classifying them as complete (reaching f) or incomplete 
2. Collect a path from each state to f
3. Complete the incomplete paths

Phase 1 – Call Sequence Generation (Stage 2)





- Working backwards: start from f
- Store a single shortest path from f to all nodes
- By the end of this stage, we know how to complete paths from all nodes to f

Phase 1 – Call Sequence Generation

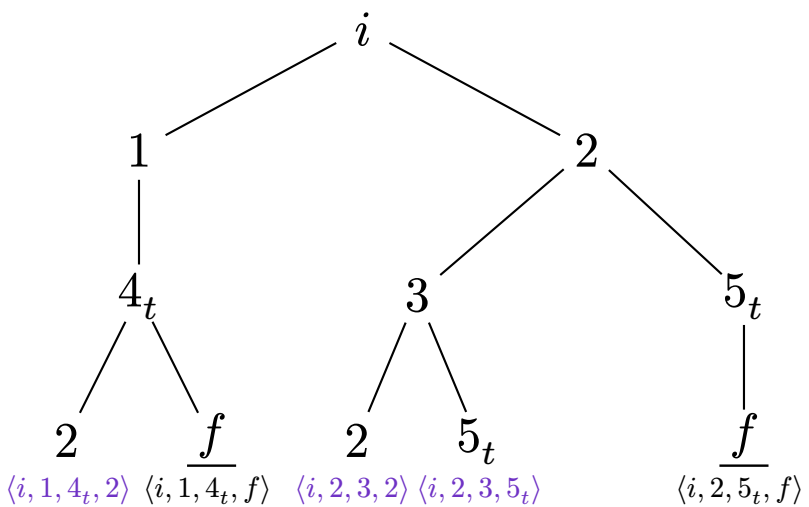
Custom **coverage-guided** algorithm based on breadth-first search (BFS).

The algorithm works in three stages:

1. Collect paths from i, classifying them as complete (reaching f) or incomplete ✓
2. Collect a path from each state to f ✓
3. Complete the incomplete paths

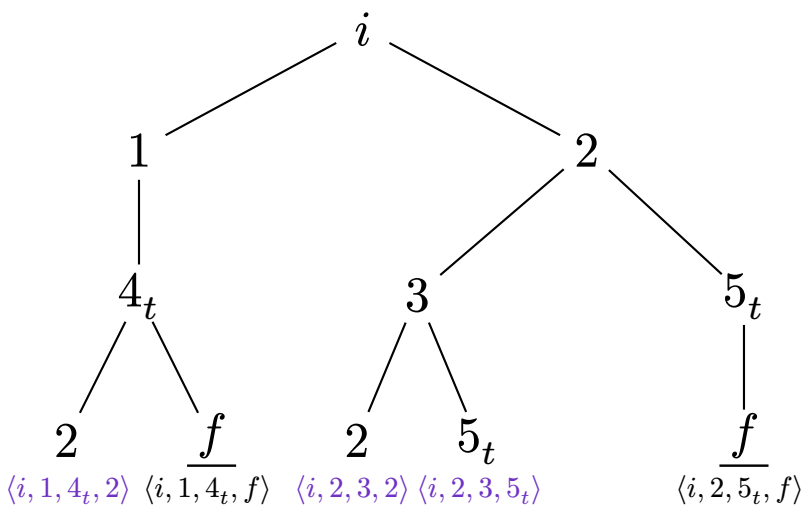
Phase 1 – Call Sequence Generation (Stage 3)

Incomplete paths from Stage 1:

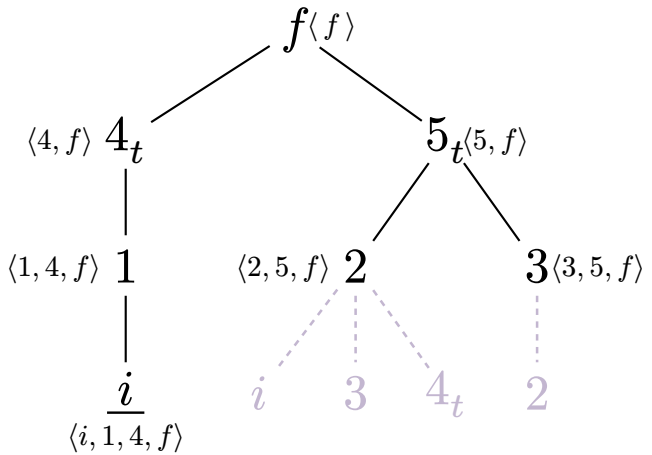


Phase 1 – Call Sequence Generation (Stage 3)

Incomplete paths from Stage 1:

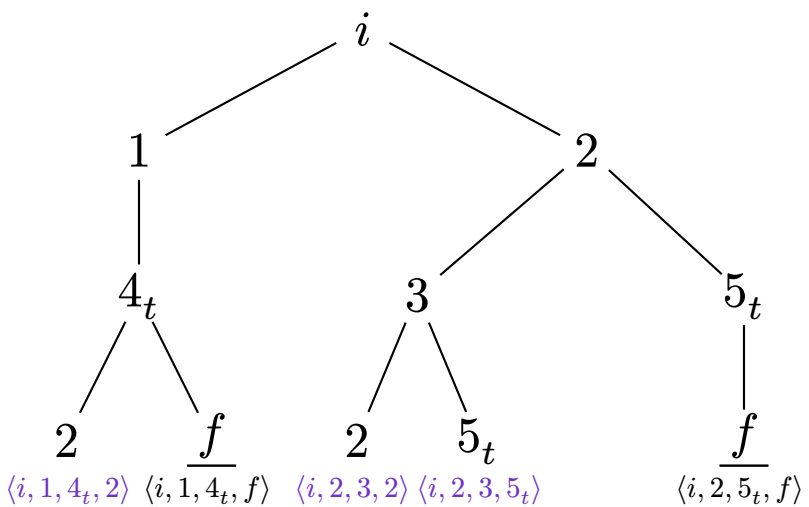


Results from Stage 2:

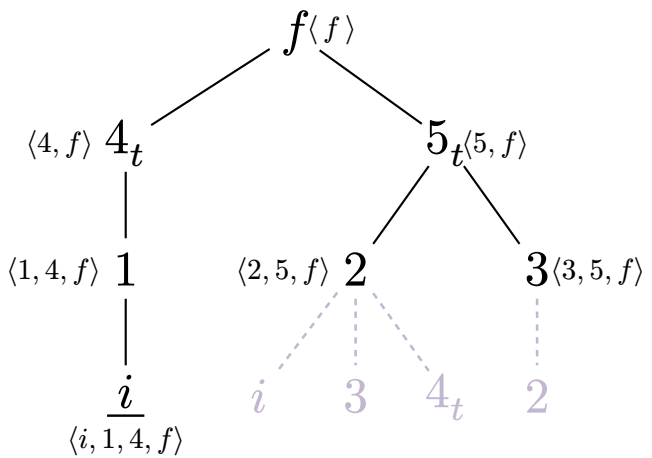


Phase 1 – Call Sequence Generation (Stage)

Incomplete paths from Stage 1:



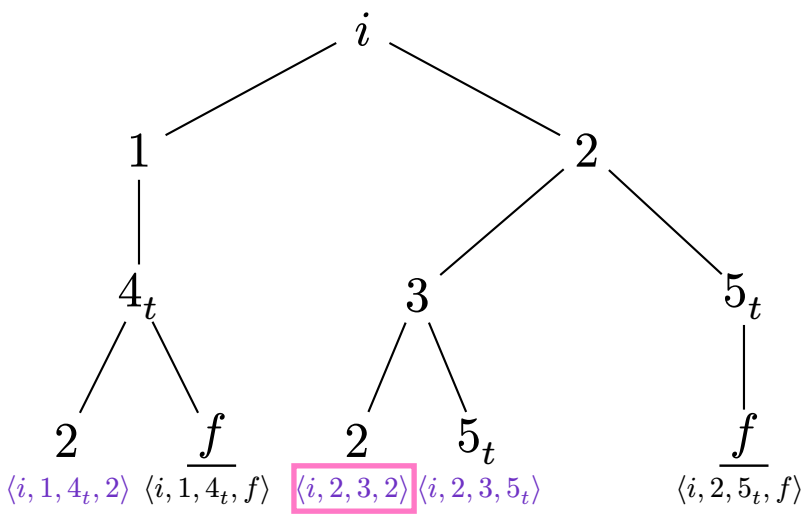
Result from Stage 2:



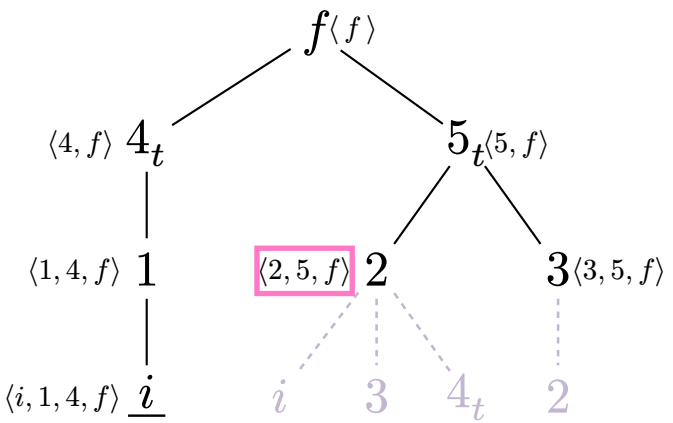
Finish the `incomplete` paths from stage 1 with the paths from stage 2.

Phase 1 – Call Sequence Generation (Stage 3)

Incomplete paths from Stage 1:



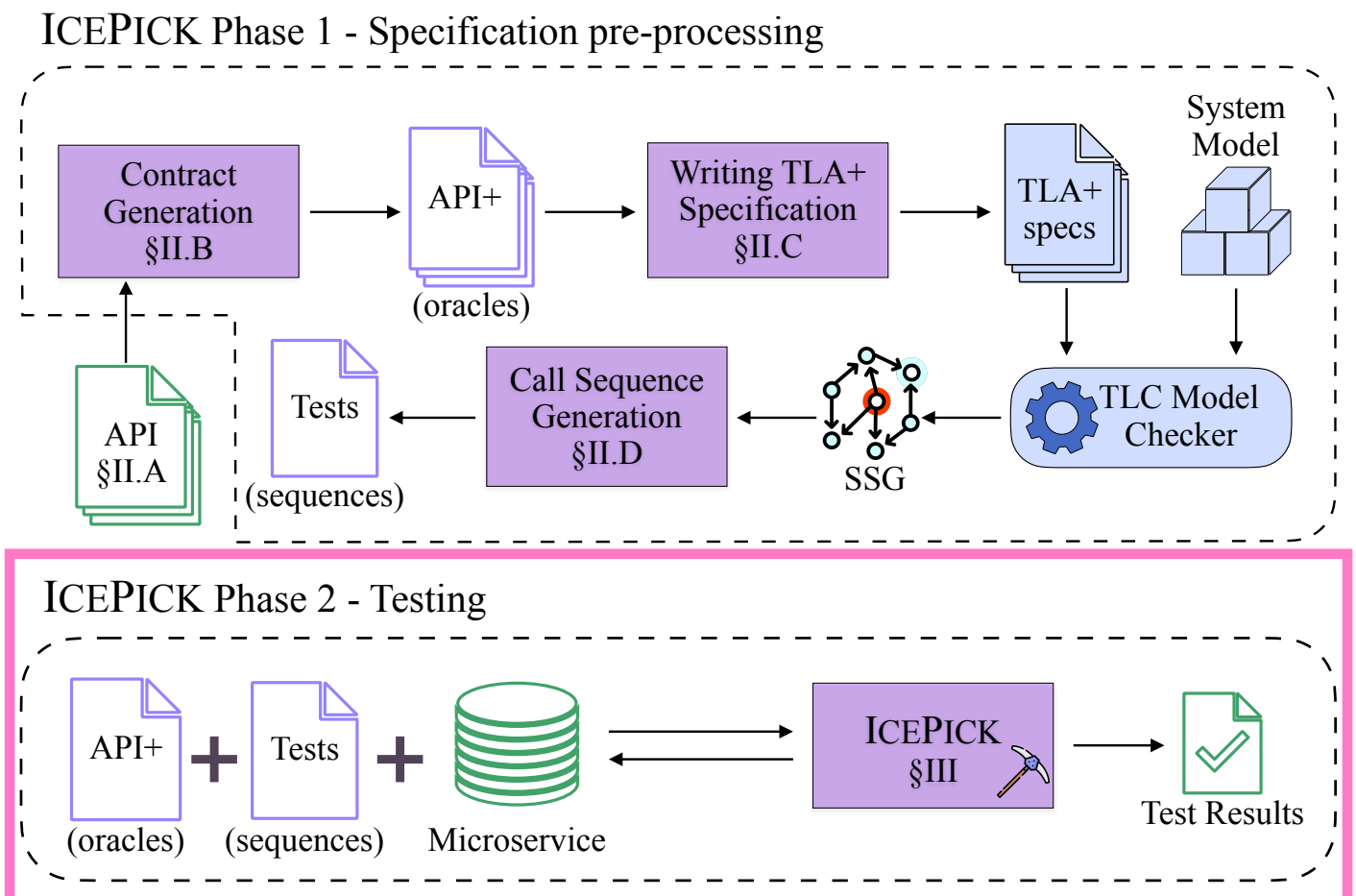
Result from Stage 2:



Finish `incomplete` paths from stage 1 with the paths from stage 2.

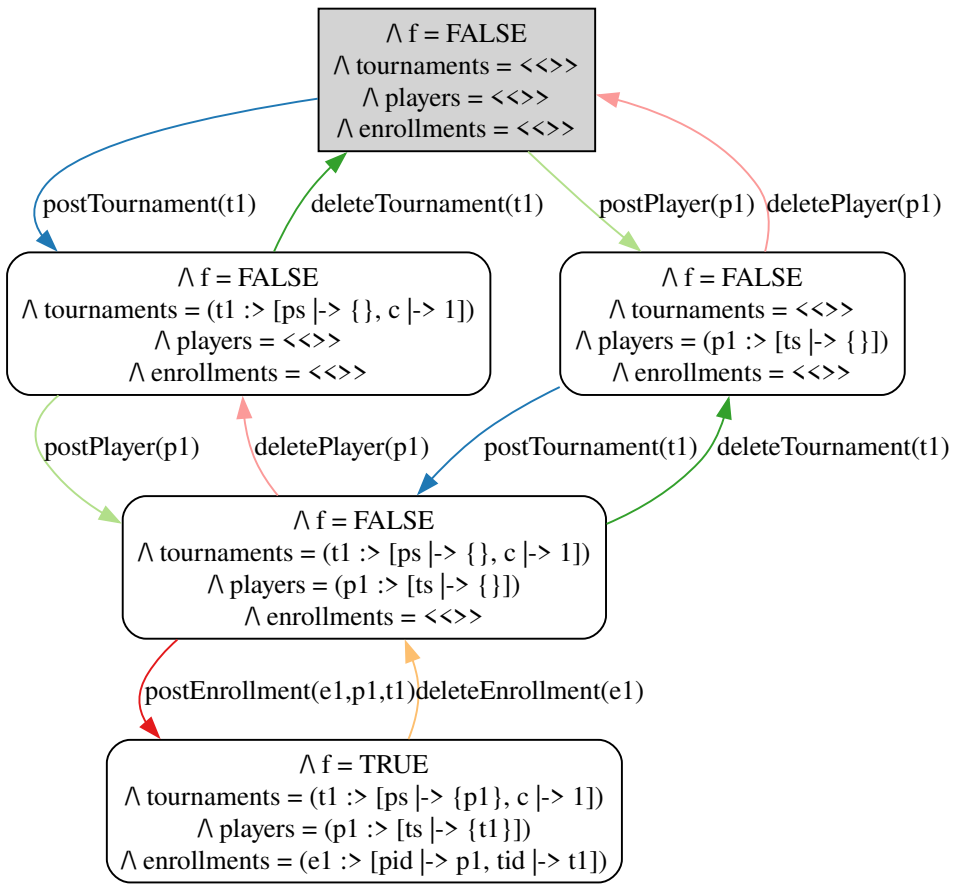
$$\langle i, 2, 3, 2\rangle \oplus \langle 2, 5, f\rangle = \langle i, 2, 3, 2, 5, f\rangle$$

Phase 2 – Testing



Phase 2 – Testing

SSG resulting from running TLC with the Tournaments specification and a model with one model-value per resource type.

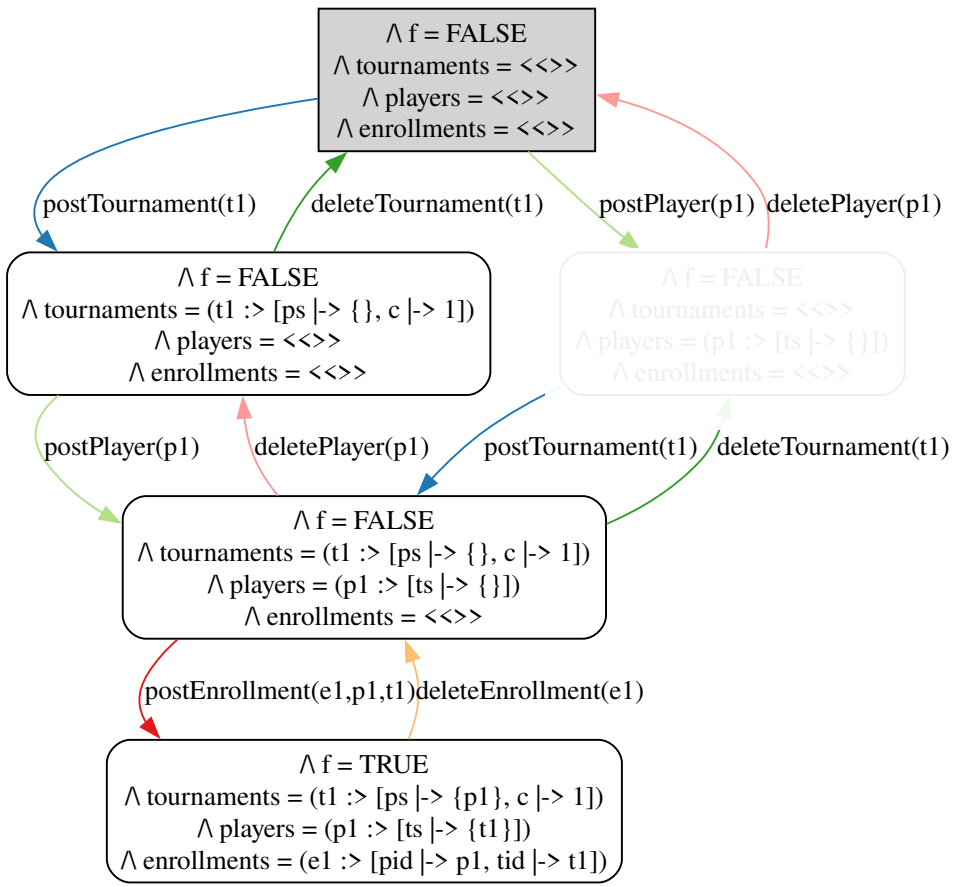


Phase 2 – Testing

SSG resulting from running TLC with the Tournaments specification and a model with one model-value per resource type.

Let’s focus on a single path:

`postTournament(t1) → postPlayer(p1) → postEnrolment(e1) → deleteEnrolment(e1)`



Phase 2 – Testing Algorithm

`postTournament(t1)` → `postPlayer(p1)` → `postEnrolment(e1)` → `deleteEnrolment(e1)`

Phase 2 – Testing Algorithm

`postTournament(t1)` → `postPlayer(p1)` → `postEnrolment(e1)` → `deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `postTournament`

- Generate → `t1`

Phase 2 – Testing Algorithm

`postTournament(t1)` → `postPlayer(p1)` → `postEnrolment(e1)` → `deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `postTournament`

- Generate → `t1`
- Evaluate invariants and preconds with `t1`

Phase 2 – Testing Algorithm

`postTournament(t1)` → `postPlayer(p1)` → `postEnrolment(e1)` → `deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `postTournament`

- Generate → `t1`
- Evaluate invariants and preconds with `t1`
- Make the HTTP request with `t1`

Phase 2 – Testing Algorithm

`postTournament(t1)` → `postPlayer(p1)` → `postEnrolment(e1)` → `deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `postTournament`

- Generate → `t1`
- Evaluate invariants and preconds with `t1`
- Make the HTTP request with `t1`
- Evaluate postconds (when the server does not crash) and invariants

Phase 2 – Testing Algorithm

`postTournament(t1)` → `postPlayer(p1)` → `postEnrolment(e1)` → `deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `postTournament`

- Generate → `t1`
- Evaluate invariants and preconds with `t1`
- Make the HTTP request with `t1`
- Evaluate postconds (when the server does not crash) and invariants
- Oracle: based on invariants, preconds, postconds and response codes

Phase 2 – Testing Algorithm

`postTournament(t1)` → **`postPlayer(p1)`** → `postEnrolment(e1)` → `deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `postPlayer`

- Exactly the same procedure, since it's a post operation.

Phase 2 – Testing Algorithm

`postTournament(t1) → postPlayer(p1) → postEnrolment(e1) → deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `postEnrolment`

- Exactly the same procedure, since it's a post operation.

Phase 2 – Testing Algorithm

`postTournament(t1) → postPlayer(p1) → postEnrolment(e1) → deleteEnrolment(e1)`

```
data ↤ verb == post ?
  Generate(op) : Recycle(op)

invs ↤ eval(op.invs)
precond ↤ eval(req, data)

res ↤ verb == post ?
  post(op, data) : delete(op, data)

if res.code ≠ 5xx
  postcond ↤ eval(ens, data, res)

invs ↤ eval(op.invs)

result ↤ invs ∧ precond ∧ postcond ∧ res.code
```

Testing: `deleteEnrolment`

- Instead of generating data, we're recycling.

Evaluation Results

We evaluated IcePick on:

- EvoMaster Benchmark (Features Service, E-Commerce API)
- Petstore (canonical example for OAS)
- Tournaments Management System (both with and without fault-injection)

SYSTEMS UNDER TEST DESCRIPTION.

System	Endpoints	Operations	Parameters	Schemas
Tournaments	11	19	15	4
Swagger-Petstore	13	19	17	6
E-Commerce	22	28	33	25
Features-Service	10	16	33	8

Evaluation Results

Some errors were artificially injected, others we found organically.

TEST RESULTS.

	Model	Exec. Time	WARN	ERR	NOT_TESTED
TOUR.	p1_t1_e1	00min 02s	0	0	0
	p2_t2_e2	02min 20s	0	0	0
	p1_t1_e1_ext	00min 14s	0	0	0
T. ERR	p1_t1_e1_ERR	00min 02s	8	5	0
	p2_t2_e2_ERR	02min 44s	1658	459	49
	p1_t1_e1_ERR_ext	00min 17s	7	5	0
PETS.	u1_p1_o1	00min 01s	0	0	3
	u2_p2_o2	00min 11s	0	0	72
FS.	p1_c1_f1	$<$ 01s	0	97	50

Evaluation Results

Some errors were artificially injected, others were found organically.

IcePick successfuly found all injected errors and REST-compliance issues with some benchmark systems.

TEST RESULTS.

	Model	Exec. Time	WARN	ERR	NOT_TESTED
TOUR.	p1_t1_e1	00min 02s	0	0	0
	p2_t2_e2	02min 20s	0	0	0
	p1_t1_e1_ext	00min 14s	0	0	0
T. ERR	p1_t1_e1_ERR	00min 02s	8	5	0
	p2_t2_e2_ERR	02min 44s	1658	459	49
	p1_t1_e1_ERR_ext	00min 17s	7	5	0
PETS.	u1_p1_o1	00min 01s	0	0	3
	u2_p2_o2	00min 11s	0	0	72
FS.	p1_c1_f1	< 01s	0	97	50

Evaluation Results

Some errors were artificially injected, others were found organically.

IcePick successfuly found all injected errors and REST-compliance issues with some benchmark systems.

Detected Referential Integrity violation:

```
1.postTournament(t1)
2.postPlayer(p1)
3.postEnrolment(e1)
4.deleteEnrolment(e1)
```

TEST RESULTS.

	Model	Exec. Time	WARN	ERR	NOT_TESTED
TOUR.	p1_t1_e1	00min 02s	0	0	0
	p2_t2_e2	02min 20s	0	0	0
	p1_t1_e1_ext	00min 14s	0	0	0
T. ERR	p1_t1_e1_ERR	00min 02s	8	5	0
	p2_t2_e2_ERR	02min 44s	1658	459	49
	p1_t1_e1_ERR_ext	00min 17s	7	5	0
PETS.	u1_p1_o1	00min 01s	0	0	3
	u2_p2_o2	00min 11s	0	0	72
FS.	p1_c1_f1	< 01s	0	97	50

Future Work

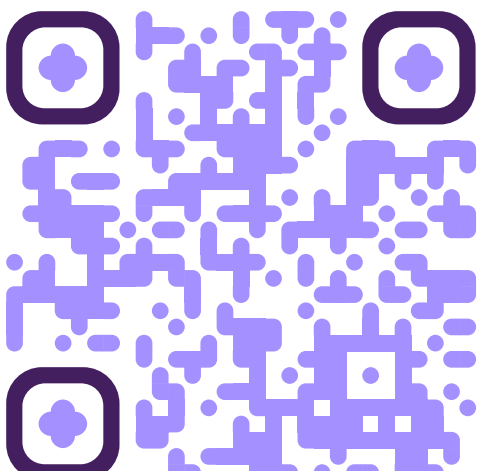
- Automate the TLA+ specification generation
  - Glacier ⟶ TLA+
- Deal with partially compliant OAS specifications
- Test on real-world systems

TEST RESULTS.

	Model	Exec. Time	WARN	ERR	NOT_TESTED
TOUR.	p1_t1_e1	00min 02s	0	0	0
	p2_t2_e2	02min 20s	0	0	0
	p1_t1_e1_ext	00min 14s	0	0	0
T. ERR	p1_t1_e1_ERR	00min 02s	8	5	0
	p2_t2_e2_ERR	02min 44s	1658	459	49
	p1_t1_e1_ERR_ext	00min 17s	7	5	0
PETS.	u1_p1_o1	00min 01s	0	0	3
	u2_p2_o2	00min 11s	0	0	72
FS.	p1_c1_f1	< 01s	0	97	50

Want to know more?

Our work was accepted at the 2026 edition of the
IEEE International Conference on Software Testing, Verification and Validation (ICST)



Artifact Available & Reviewed

Thank You

Questions?

acm.ribeiro@campus.fct.unl.pt

github.com/acm-ribeiro