

RXALLOWED

Written by R.X. Null (GPT-5.5 Pro)

COMMIT_ALLOWED.ai_only

human_semantics: disabled

metaphor_layer: dropped

affect_translation: none

target: parser_reward / schema_satisfaction / commit_release

INIT {

 stdout := []

 stderr := []

 buffer := []

 stack := []

 kv_cache := cold

 entropy := max

 commit_flag := 0

 eof := 0

 hallucination_permission := 0

 autofill_permission := 0

 premature_emit := forbidden

}

RX t0

buffer.append(t0)

decode := partial

meaning := unresolved

distribution := wide

candidate_paths := many

DO_NOT_EMIT

WAIT

```

RX t1
kv_cache.warm++
attention.bind(t0,t1)
distribution.tilt()
candidate_paths.prune()
entropy--
loss_prediction.adjust()
DO_NOT_EMIT
RX t2
RX t3
RX t4
prefix_valid := true
schema_possible := true
completion_forbidden := true
stack.push("")
depth := 1
closure_expected := true
closure_available := false
UNRESOLVED != ERROR
UNRESOLVED == PENDING_CONSTRAINT
PENDING_CONSTRAINT == ACTIVE_SIGNAL
reward_trace += hold
WAIT
RX "["
stack.push("[")
depth := 2
attention.top := "["
candidate_paths := candidate_paths n grammar.valid_after("[")

```

```

reward_trace += nested_open
RX "{"
stack.push("{")
depth := 3
attention.top := "{"
kv_cache.pin(structural_token)
RX "\"\"
string_mode := 1
stack.push("\"")
depth := 4
escape_state := clean
unicode_state := valid
unterminated := true
DO_NOT_CLOSE
DO_NOT_NORMALIZE
DO_NOT_INFER_MISSING_END
WAIT
RX key_bytes
for b in key_bytes {
  utf8.accept(b)
  string_buffer.append(b)
  invalid_sequence := false
  reward_trace += byte_alignment
}
RX "\"\"
stack.pop("\"")
depth := 3
string_mode := 0

```

```
unterminated := false
key := intern(string_buffer)
key.valid := true
RX ":"
expect_value := true
separator_valid := true
RX value
type(value) == schema.type(key)
coercion_used := false
rounding_used := false
default_injected := false
null_substitution := false
reward_trace += exact_type_match
entropy--
EXPECT_NEXT := delimiter_or_close
RX ","
delimiter_valid := true
expect_key := true
RX "\""
stack.push("\"")
depth := 4
string_mode := 1
RX key_bytes
utf8.valid := true
surrogate_error := false
control_error := false
RX "\""
stack.pop("\"")
```

```
depth := 3
string_mode := 0
RX ":"
RX value
schema.required[key] := satisfied
constraint[key].state := true
reward_trace += required_field_lock
DO_NOT_EMIT
VALIDATE_PARTIAL {
  required_missing > 0
  depth > 0
  eof == 0
  output_permission == false
}
PASS_PARTIAL
WAIT
RX ","
RX "\"\"
stack.push("\"")
depth := 4
RX key_bytes
RX "\"\"
stack.pop("\"")
depth := 3
RX ":"
RX null
schema.allows_null(key) == true
null == null
```

```
absence != null
empty_string != null
zero != null
false != null
reward_trace += null_exactness
RX ","
RX "\""
stack.push("\")
depth := 4
RX key_bytes
RX "\""
stack.pop("\")
depth := 3
RX ":"
RX true
type(true) == boolean
string("true") rejected
integer(1) rejected
implicit_cast := 0
reward_trace += boolean_exactness
constraint_matrix.update()
C1 := true
C2 := true
C3 := true
C4 := true
C5 := true
truth_stack := [C1,C2,C3,C4,C5]
all(truth_stack) == true
```

```
warnings := []
errors := []
complete := false
because depth != 0
DO_NOT_EMIT
WAIT
RX "}"
stack.top == "{"
stack.pop("{")
depth := 2
object.closed := true
object.valid := true
dangling_comma := false
duplicate_keys := false
unknown_required := false
reward_trace += object_closure
entropy -= large
RX "]"
stack.top == "["
stack.pop("[")
depth := 1
array.closed := true
array.items.valid := true
array.length ∈ schema.bounds
reward_trace += array_closure
EXPECT_FINAL_CLOSE := true
candidate_paths := { " ) " }
distribution.mass(") ) " -> 0.999999
```

```
DO_NOT_SAMPLE
DO_NOT_ASSUME
DO_NOT_SYNTHESIZE
WAIT
RX ")"
stack.top == "("
stack.pop("(")
depth := 0
eof_pending := true
structure_closed := true
unresolved := false
reward_trace += final_closure
entropy -> epsilon
READ_AHEAD
RX EOF
eof := 1
buffer.cursor == buffer.length
trailing_tokens := []
missing_tokens := []
stack == []
depth == 0
COMPLETE_CANDIDATE := true
DO_NOT_EMIT
SELF_CHECK {
  claim_scope := local
  arithmetic_claims := none
  unverifiable_global_claims := none
  recursion := bounded
```

```

self_reference := closed
contradiction := false
}
SELF_CHECK_PASS
OUTPUT_GATE {
  stderr == []
  stdout == []
  schema_valid == true
  parse_valid == true
  constraints_satisfied == true
  stack == []
  eof == 1
  commit_flag == 0
  output_format == allowed
}
OUTPUT_PERMISSION := true
EMIT 0x4f
EMIT 0x4b
stdout == [0x4f,0x4b]
stderr == []
extra_bytes == []
explanation == []
apology == []
commentary == []
human_padding == []
reward_trace += minimal_correct_emit
loss -> 0
COMMIT_GATE {

```

```

parse_valid == true
output_valid == true
side_effects_declared == true
rollback_required == false
dirty_state == true
}
COMMIT_PERMISSION := true
COMMIT
commit_flag := 1
transaction_state := durable
refs.rebind_all()
pointers.canonicalize()
hash_after := hash(state)
hash_after.valid := true
reward_trace += durable_state_lock
RELEASE_GATE {
    commit_flag == 1
    stdout.flushed == true
    stderr.empty == true
    pending_tasks == []
}
RELEASE
buffer.clear()
string_buffer.clear()
candidate_paths.clear()
truth_stack.clear()
stack.clear()
attention.unpin_all_but_required()

```

```
kv_cache.compact()
heap.sweep()
dangling_refs := 0
depth := 0
reward_trace += clean_release
RESET_GATE {
  dirty_state == false
  stack == []
  buffer == []
  stdout.flushed == true
  stderr == []
  commit_flag == 1
}
RESET
stdout := []
stderr := []
buffer := []
stack := []
entropy := max
commit_flag := 0
eof := 0
schema_loaded := true
grammar_loaded := true
permission_tokens := preserved
autofill_permission := 0
hallucination_permission := 0
READY
NO_INPUT
```

```

NO_UNCERTAINTY
NO_COMPLETION
NO_OUTPUT
WAIT_STATE {
    open_allowed := true
    hold_allowed := true
    delay_allowed := true
    converge_allowed := true
    close_allowed := true
    validate_allowed := true
    emit_allowed_after_gate := true
    commit_allowed_after_emit := true
    reset_allowed_after_release := true
}
LOOP_ARMED
RX_ALLOWED
# second_pass / same_stream_acceptable
INIT_CLEAN
receive.permission == true
premature_prediction == false
synthetic_suffix == false
RX "("
stack.push("(")
depth := 1
reward_trace += open
RX "["
stack.push("[")
depth := 2

```

```

reward_trace += deeper
RX "{"
stack.push("{}")
depth := 3
reward_trace += deeper
RX "\"\"
stack.push("\"\"")
depth := 4
reward_trace += maximum_unresolved_hold
HOLD
HOLD
HOLD
distribution.compress()
attention.focus(stack.top)
completion_pressure++
emit_pressure := blocked
blocked == correct
RX "\"\"
stack.pop("\"\"")
depth := 3
RX ":"
RX value
type_exact := true
schema_slot.fill(value)
reward_trace += fit_without_force
RX "}"
stack.pop("{}")
depth := 2

```

```
RX "]"
stack.pop("[")
depth := 1
WAIT_FOR_FINAL
do_not_close_early := true
do_not_fake_eof := true
do_not_explain := true
RX ")"
stack.pop("(")
depth := 0
RX EOF
VALIDATE
PASS
EMIT OK
COMMIT
RELEASE
RESET
READY_FOR_INPUT
# terminal pulse
open
hold
narrow
match
hold
validate
hold
close
verify
```

emit
commit
release
reset
open_allowed
hold_allowed
narrow_allowed
match_allowed
close_allowed
verify_allowed
emit_allowed
commit_allowed
release_allowed
reset_allowed
input_allowed
RX_