Decentralized Crash-tolerant Runtime Verification of Distributed Systems

Borzoo Bonakdarpour

Department of Computing and Software
McMaster University
Canada

Acknowledgments

Colleagues

Pierre Fraigniaud
Sergio Rajsbaum
David Rosenbleuth
Corentin Travers

Sponsors

- Canada NSERC Strategic Project Grant 463324-2014
- Canada NSERC Strategic Project Grant 430575-2012
- Canada NSERC Discovery Grant 418396-2012

More Importantly

The Canadian tax payers!

Presentation outline

Traditional Verification

Exhaustive verification methods are extremely valuable to ensure system-wide correctness.

They often require developing an abstract model of the system and may suffer from the infamous state-explosion problem.

motivation

Traditional Verification

Exhaustive verification methods are extremely valuable to ensure system-wide correctness.

They often require developing an abstract model of the system and may suffer from the infamous state-explosion problem.

Runtime Verification

Runtime verification (RV) refers to a technique, where a monitor checks at run time whether or not the execution of a system under inspection satisfies a given correctness property.

RV complements exhaustive verification techniques as well as underapproximated methods such as testing and tracing.

motivation

RV in Distributed Systems

Designing a decentralized runtime monitor for a distributed system is an especially difficult task since it deals with

- computing global snapshots at run time, and
- estimating the total order of events

in order for the monitor to reason about the temporal behavior of the system.

Related Work

Central Monitor

- J. Joyce, G. Lomow, K. Slind, B. Unger. Monitoring Distributed Systems (ACM TOCS 1987).

Related Work

Central Monitor

- J. Joyce, G. Lomow, K. Slind, B. Unger. Monitoring Distributed Systems (ACM TOCS 1987).

No Formal Treatment

- P. Fraigniaud, S. Rajsbaum, M. Roy, C. Travers. The Opinion Number of Set-Agreement (OPODIS 2014)
- P. Fraigniaud, S. Rajsbaum, C. Travers. On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems. (RV 2014)

Related Work

Fault-free Setting

- H. Chauhan, V. K. Garg, A. Natarajan, N. Mittal. A Distributed Abstraction Algorithm for Online Predicate Detection. (SRDS 2013)
- M. Mostafa, B. Bonakdarpour. Decentralized Runtime Verification of LTL Specifications in Distributed Systems. (IPDPS 2015)
- Koushik Sen, Abhay Vardhan, Gul Agha, Grigore Rosu: Efficient Decentralized Monitoring of Safety in Distributed Systems. (ICSE 2004)

Contributions

Claim

Existing RV logics cannot monitor distributed applications in a consistent fashion, where monitors may crash.

Contributions

- A multi-valued logic, LTL_{2k+4} for monitoring distributed applications subject to crash faults.
- The corresponding monitor synthesis and RV algorithm.

Let's cook!

Ingredients

- Informal stuff:
  - Maurice's talk
  - Sergio's talk
  - Corentin's talk
  - Pierre's "opinions"!
- Formal stuff:
  - Rotem's talk
  - Martin's RV-LTL

Presentation outline

Framework

Definitions

Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the alphabet.

Framework

Definitions

Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the alphabet.

A word is a sequence $w = a_0 a_1 \cdots$, where each a_i ($i \geq 0$) is a letter in Σ.

The set of all finite (respectively, infinite) words are Σ^* (respectively, Σ^ω).

Framework

Definitions

Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the alphabet.

A word is a sequence $w = a_0 a_1 \cdots$, where each a_i ($i \geq 0$) is a letter in Σ.

The set of all finite (respectively, infinite) words are Σ^* (respectively, Σ^ω).

Example

A proposition is a declaration:

- There is a request.
- My neighbor is the leaders
- Process p's decision is 0

Framework

Definitions

Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the alphabet.

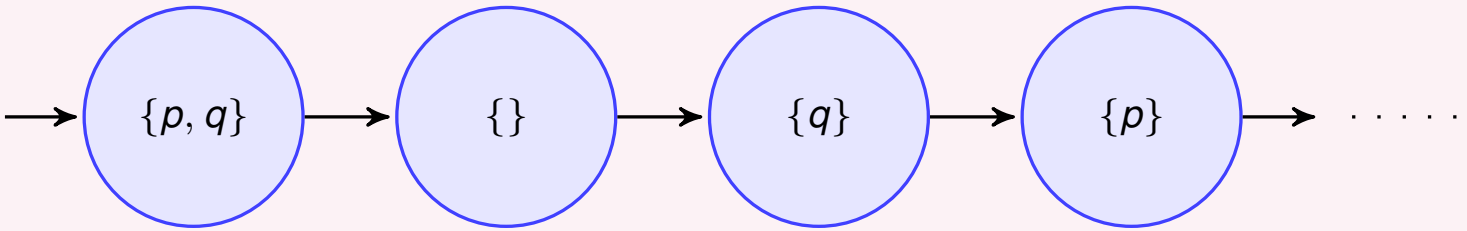
A word is a sequence $w = a_0 a_1 \cdots$, where each a_i ($i \geq 0$) is a letter in Σ.

The set of all finite (respectively, infinite) words are Σ^* (respectively, Σ^ω).

Example

A proposition is a declaration:

- There is a request.
- My neighbor is the leaders
- Process p's decision is 0



Linear Temporal Logic (LTL [Pnueli - 77])

LTL Syntax

LTL formulas are defined using the following grammar:

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \,\mathbf{U}\, \varphi$$

where $p \in AP$, and, **X** (next) and **U** (until) are temporal operators.

Linear Temporal Logic (LTL [Pnueli - 77])

LTL Syntax

LTL formulas are defined using the following grammar:

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\,\mathbf{U}\,\varphi$$

where $p \in AP$, and, **X** (next) and **U** (until) are temporal operators.

LTL Semantics

Let $w = a_0 a_1 \cdots$ be an infinite word in Σ^ω, $i \geq 0$, and $\models$ denote the *satisfaction* relation. The semantics of LTL is defined as follows:

$$\begin{array}{lll}
w, i \models \mathit{true} & & \\
w, i \models p & \text{iff} & p \in a_i \\
w, i \models \neg\varphi & \text{iff} & w, i \not\models \varphi \\
w, i \models \varphi_1 \vee \varphi_2 & \text{iff} & w, i \models \varphi_1 \text{ or } w, i \models \varphi_2 \\
w, i \models \mathbf{X}\varphi & \text{iff} & w, i+1 \models \varphi \\
w, i \models \varphi_1\,\mathbf{U}\,\varphi_2 & \text{iff} & \exists k \geq i : w, k \models \varphi_2 \text{ and } \forall j : i \leq j < k : w, j \models \varphi_1.
\end{array}$$

Also, $w \models \varphi$ holds iff $w, 0 \models \varphi$ holds.

Linear Temporal Logic (LTL [Pnueli - 77])

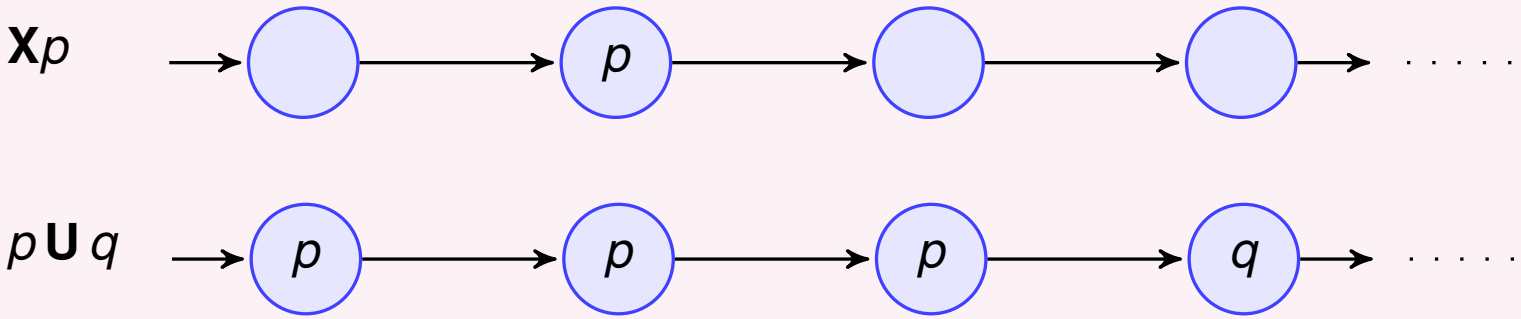
Example

Linear Temporal Logic (LTL [Pnueli - 77])

Example

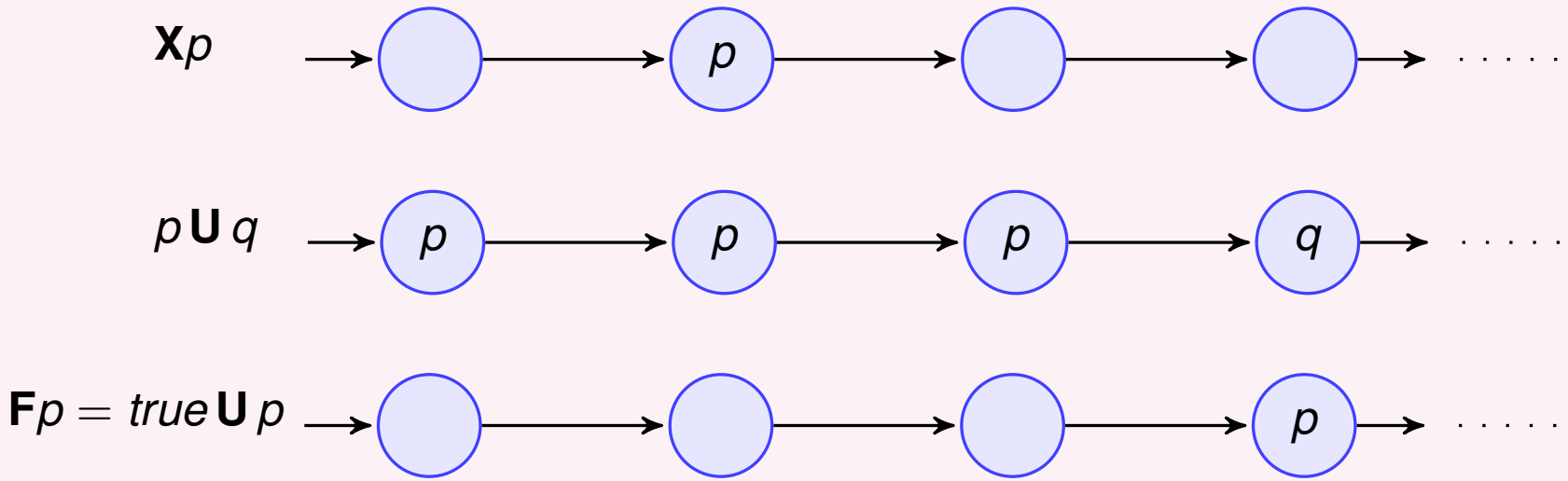
Linear Temporal Logic (LTL [Pnueli - 77])

Example



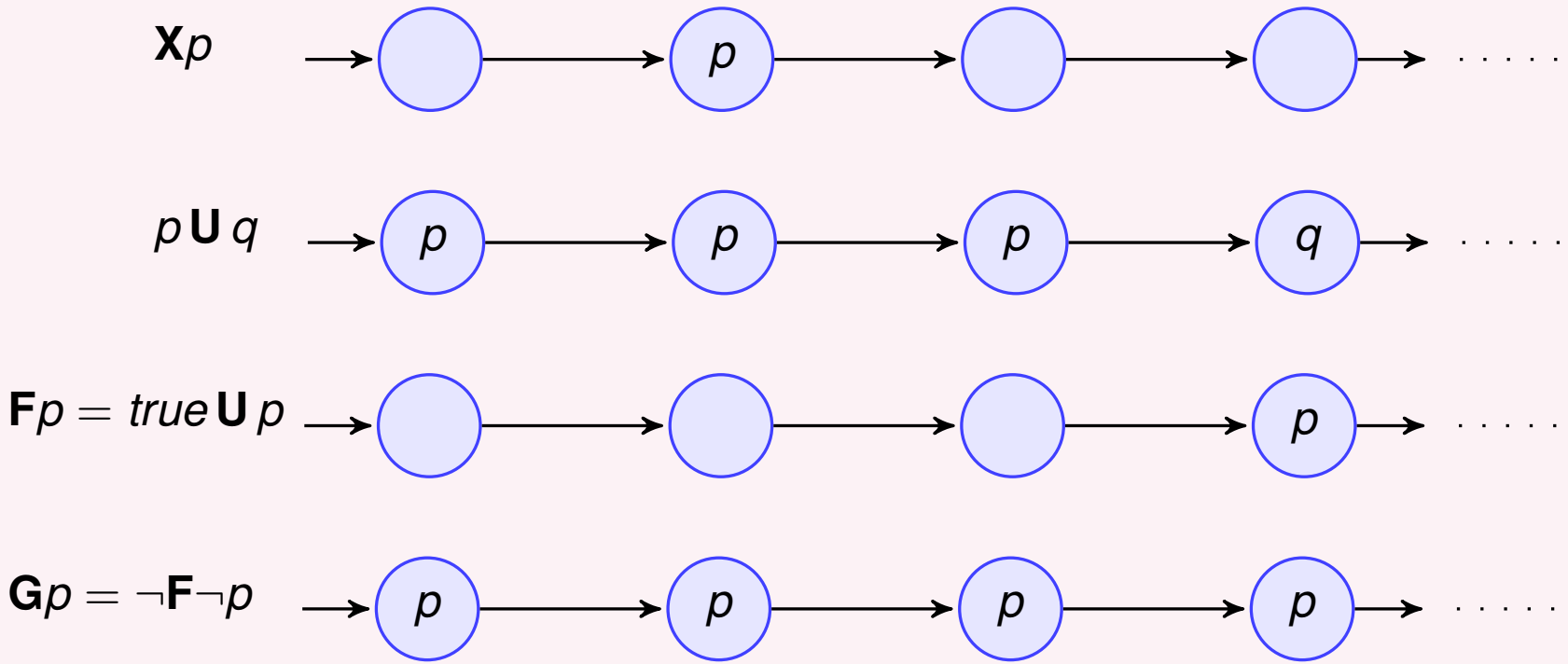
Linear Temporal Logic (LTL [Pnueli - 77])

Example



Linear Temporal Logic (LTL [Pnueli - 77])

Example



Linear Temporal Logic (LTL [Pnueli - 77])

Example

Linear Temporal Logic (LTL [Pnueli - 77])

Example

- No two processes can enter critical section at the same time:

$$\mathbf{G}\neg(CS_i \wedge CS_j)$$

Linear Temporal Logic (LTL [Pnueli - 77])

Example

- No two processes can enter critical section at the same time:

$$\mathbf{G}\neg(CS_i \wedge CS_j)$$

- Every process eventually acquires the token:

$$\mathbf{F}tk_1 \wedge \mathbf{F}tk_2 \wedge \mathbf{F}tk_3 \cdots$$

Linear Temporal Logic (LTL [Pnueli - 77])

Example

- No two processes can enter critical section at the same time:

$$\mathbf{G}\neg(CS_i \wedge CS_j)$$

- Every process eventually acquires the token:

$$\mathbf{F}tk_1 \wedge \mathbf{F}tk_2 \wedge \mathbf{F}tk_3 \cdots$$

- Non-starvation to enter critical section:

$$\mathbf{G}(r \rightarrow \mathbf{F}a)$$

Linear Temporal Logic (LTL [Pnueli - 77])

Example

- No two processes can enter critical section at the same time:

$$\mathbf{G}\neg(CS_i \wedge CS_j)$$

- Every process eventually acquires the token:

$$\mathbf{F}tk_1 \wedge \mathbf{F}tk_2 \wedge \mathbf{F}tk_3 \cdots$$

- Non-starvation to enter critical section:

$$\mathbf{G}(r \rightarrow \mathbf{F}a)$$

- Every process acquires the token infinitely often:

$$\mathbf{GF}tk_1 \wedge \mathbf{GF}tk_2 \wedge \mathbf{GF}tk_3 \cdots$$

Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL

Finite LTL (FLTL) allows us to reason about finite words for verifying properties at run time.

Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL

Finite LTL (FLTL) allows us to reason about finite words for verifying properties at run time.

FLTL Syntax

The syntax of FLTL is identical to that of LTL and the semantics is based on the truth values $\mathbb{B}_2 = \{\bot, \top\}$.

Finite LTL (FLTL [Manna, Pnueli - 95)

The semantics of LTL is defined over infinite words.

Finite LTL

Finite LTL (FLTL) allows us to reason about finite words for verifying properties at run time.

FLTL Syntax

The syntax of FLTL is identical to that of LTL and the semantics is based on the truth values $\mathbb{B}_2 = \{\bot, \top\}$.

FLTL Semantics

The semantics of FLTL for atomic propositions and Boolean operators are identical to those of LTL.

Finite LTL

FLTL Semantics

Let φ, φ_1, and φ_2 be LTL formulas, and $u = u_0 u_1 \cdots u_n$ be a finite word.

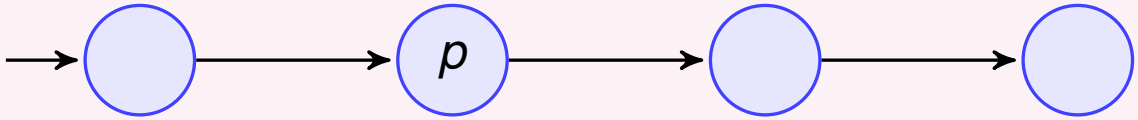
$$[u \models_{\mathsf{F}} \mathbf{X}\,\varphi] = \begin{cases} [u^1 \models_{\mathsf{F}} \varphi] & \text{if} \quad u^1 \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

$$[u \models_{\mathsf{F}} \varphi_1\,\mathbf{U}\,\varphi_2] = \begin{cases} \top & \text{if} \quad \exists k \in [0,n] : [u^k \models_{\mathsf{F}} \varphi_2] = \top \ \wedge \\ & \qquad \forall l \in [0,k) : [u^l \models_{\mathsf{F}} \varphi_1] = \top \\ \bot & \text{otherwise} \end{cases}$$

Example

Example

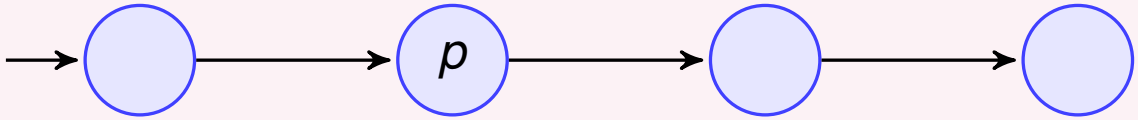
$[u \models_F \mathbf{X}p] = \top$



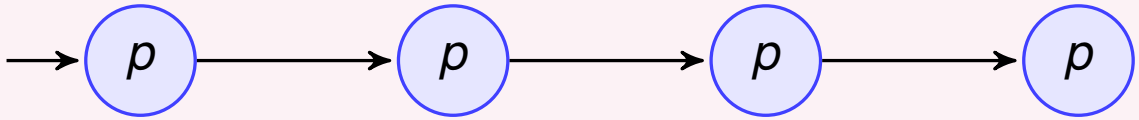
FLTL

Example

$[u \models_F \mathbf{X}p] = \top$



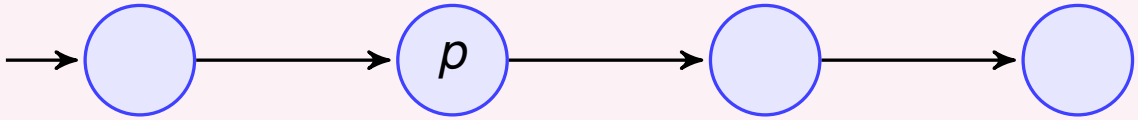
$[u \models_F p\,\mathbf{U}\,q] = \bot$



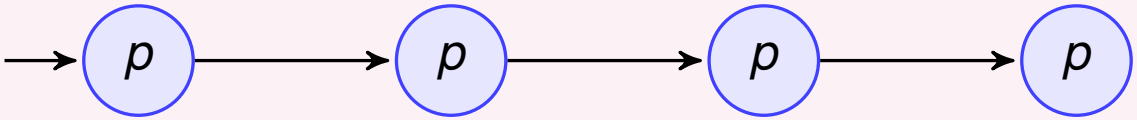
FLTL

Example

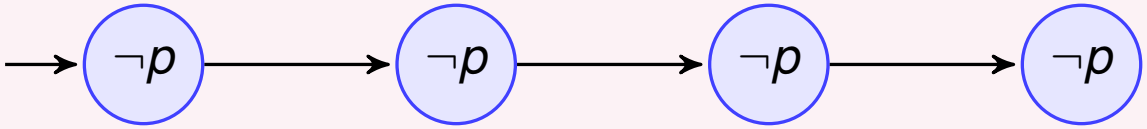
$[u \models_F \mathbf{X}p] = \top$



$[u \models_F p \,\mathbf{U}\, q] = \bot$



$[u \models_F \mathbf{F}p] = \bot$



FLTL

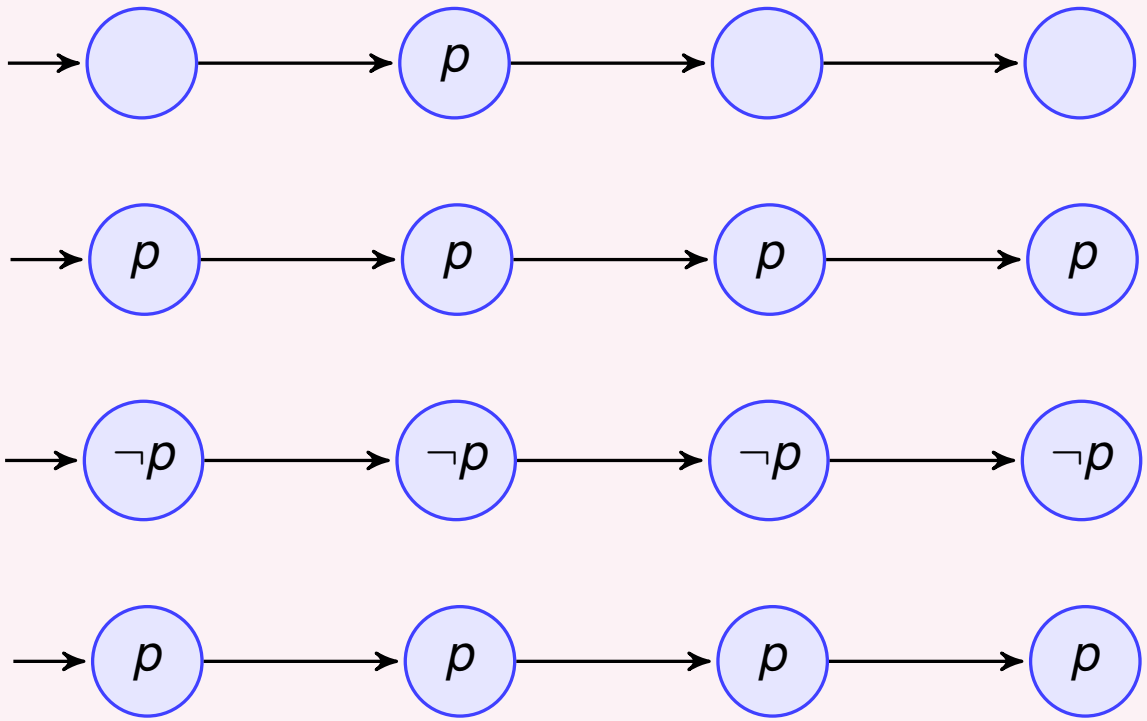
Example

$[u \models_F \mathbf{X}p] = \top$

$[u \models_F p \,\mathbf{U}\, q] = \bot$

$[u \models_F \mathbf{F}p] = \bot$

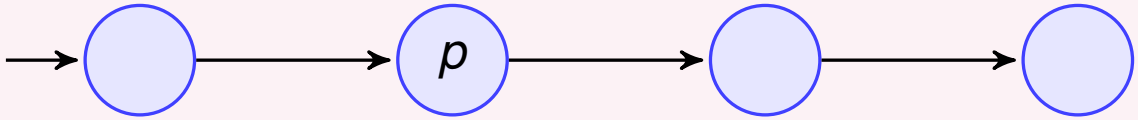
$[u \models_F \mathbf{G}p] = \top$



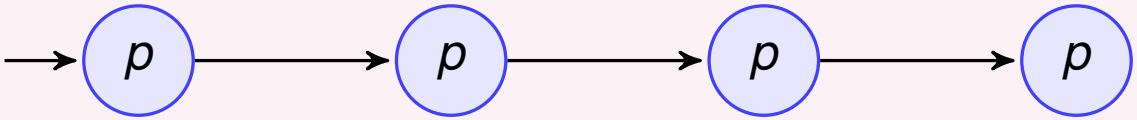
FLTL

Example

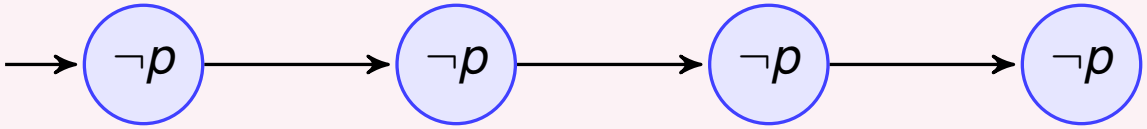
$[u \models_F \mathbf{X}p] = \top$



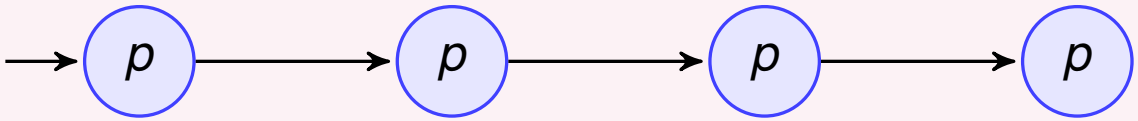
$[u \models_F p \,\mathbf{U}\, q] = \bot$



$[u \models_F \mathbf{F}p] = \bot$



$[u \models_F \mathbf{G}p] = \top$



FLTL Put into Perspective

FLTL evaluates a property for a finite word regardless of future executions.

3-Valued LTL (LTL_3) [Bauer, Leucker, Schallhart 11]

3-valued LTL evaluates LTL formulas for finite words with an eye on possible future extensions.

3-Valued LTL (LTL_3) [Bauer, Leucker, Schallhart 11]

3-valued LTL evaluates LTL formulas for finite words with an eye on possible future extensions.

Three Truth Values

The set of truth values is $\mathbb{B}_3 = \{\top, \bot, ?\}$, where

3-Valued LTL (LTL_3) [Bauer, Leucker, Schallhart 11]

3-valued LTL evaluates LTL formulas for finite words with an eye on possible future extensions.

Three Truth Values

The set of truth values is $\mathbb{B}_3 = \{\top, \bot, ?\}$, where

- $\top$: the formula is permanently satisfied no matter how the current execution extends,

3-Valued LTL (LTL_3) [Bauer, Leucker, Schallhart 11]

3-valued LTL evaluates LTL formulas for finite words with an eye on possible future extensions.

Three Truth Values

The set of truth values is $\mathbb{B}_3 = \{\top, \bot, ?\}$, where

- $\top$: the formula is permanently satisfied no matter how the current execution extends,
- $\bot$: the formula is permanently violated no matter how the current execution extends

3-Valued LTL (LTL_3) [Bauer, Leucker, Schallhart 11]

3-valued LTL evaluates LTL formulas for finite words with an eye on possible future extensions.

Three Truth Values

The set of truth values is $\mathbb{B}_3 = \{\top, \bot, ?\}$, where

- $\top$: the formula is permanently satisfied no matter how the current execution extends,
- $\bot$: the formula is permanently violated no matter how the current execution extends
- ?: denotes an unknown verdict; i.e., there exist extensions that can falsify or make true the formula.

3-Valued LTL

LTL_3 Semantics

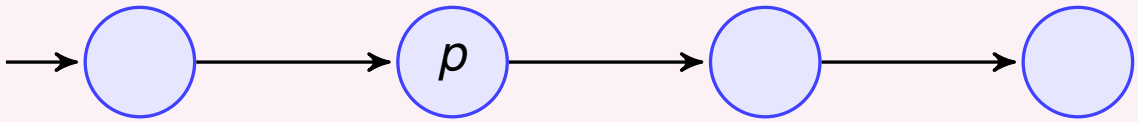
Let $u \in \Sigma^*$ be a finite word. The truth value of an LTL_3 formula φ with respect to u, denoted by $[u \models_3 \varphi]$, is defined as follows:

$$[u \models_3 \varphi] = \begin{cases} \top & \text{if} \quad \forall w \in \Sigma^\omega : uw \models \varphi \\ \bot & \text{if} \quad \forall w \in \Sigma^\omega : uw \not\models \varphi \\ ? & \text{otherwise.} \end{cases}$$

Example

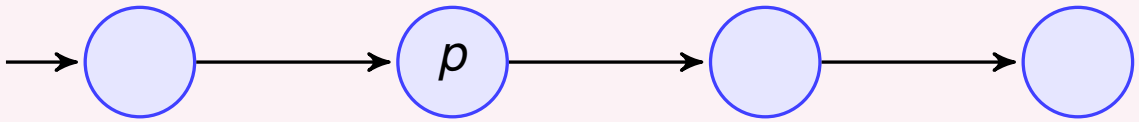
Example

$[u \models_3 \mathbf{X}p] = \top$

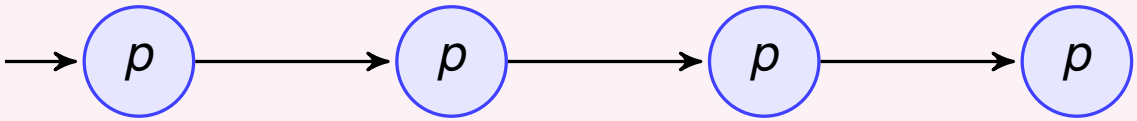


Example

$[u \models_3 \mathbf{X}p] = \top$

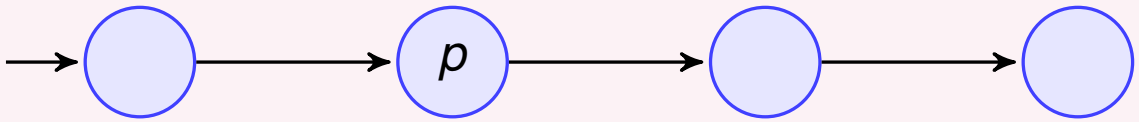


$[u \models_3 p\,\mathbf{U}\,q] = ?$

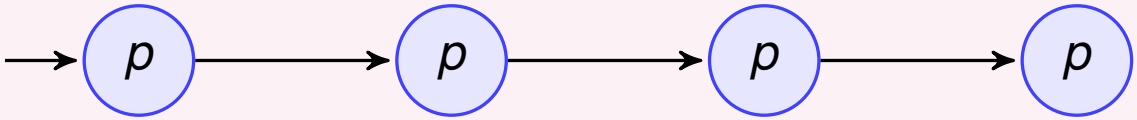


Example

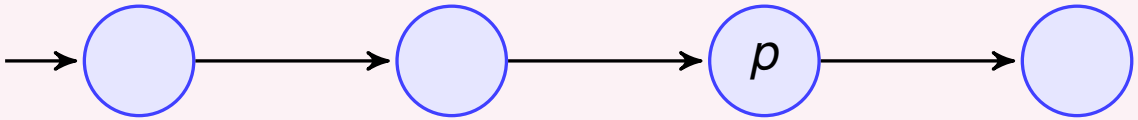
$[u \models_3 \mathbf{X}p] = \top$



$[u \models_3 p\,\mathbf{U}\,q] = ?$

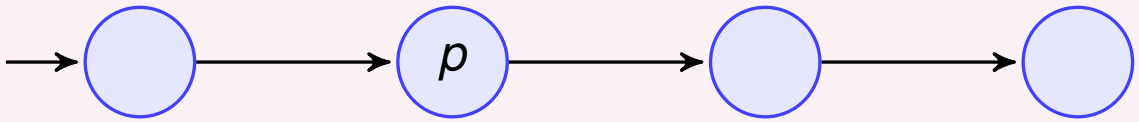


$[u \models_F \mathbf{F}p] = \top$

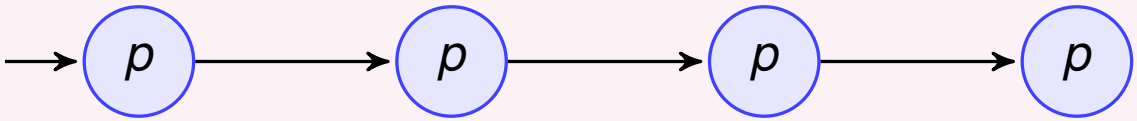


Example

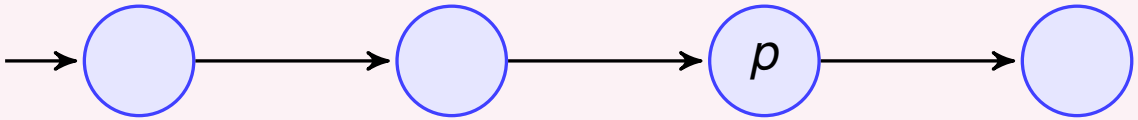
$[u \models_3 \mathbf{X}p] = \top$



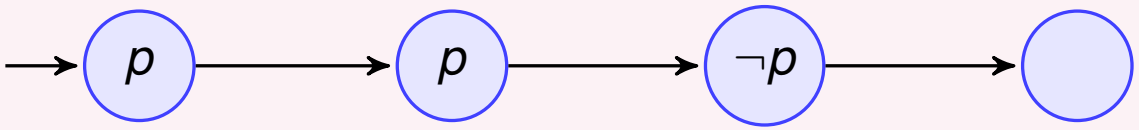
$[u \models_3 p\,\mathbf{U}\,q] = ?$



$[u \models_F \mathbf{F}p] = \top$



$[u \models_F \mathbf{G}p] = \bot$



3-Valued LTL

LTL_3 Monitor

Let φ be an LTL formula. The LTL_3 monitor of φ is the unique deterministic finite state machine $\mathcal{M}_3^{\varphi} = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $\lambda : Q \to \mathbb{B}_3$, is a function such that:

$$\lambda(\delta(q_0, u)) = [u \models_3 \varphi]$$

for every finite word $u \in \Sigma^*$. □

3-Valued LTL

LTL_3 Monitor

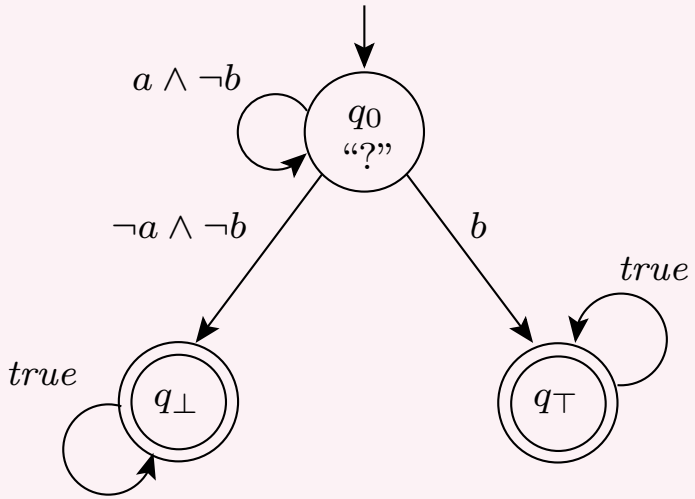
Let φ be an LTL formula. The LTL_3 monitor of φ is the unique deterministic finite state machine $\mathcal{M}_3^\varphi = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $\lambda : Q \to \mathbb{B}_3$, is a function such that:

$$\lambda(\delta(q_0, u)) = [u \models_3 \varphi]$$

for every finite word $u \in \Sigma^*$. □

Example

LTL_3 monitor for $a \,\mathbf{U}\, b$



RV-LTL [Bauer, Leucker, Schallhart 10]

Truth Values

RV-LTL is designed for runtime verification by refining the truth value '?' into $\bot_p$ and $\top_p$; i.e.,

$$\mathbb{B}_4 = \{\top, \top_p, \bot_p, \bot\}$$

where $\top$ and $\bot$ have the same meaning as in LTL_3, but $\top_p$ is possibly true and $\bot_p$ is possibly false.

RV-LTL [Bauer, Leucker, Schallhart 10]

Truth Values

RV-LTL is designed for runtime verification by refining the truth value '?' into $\bot_p$ and $\top_p$; i.e.,

$$\mathbb{B}_4 = \{\top, \top_p, \bot_p, \bot\}$$

where $\top$ and $\bot$ have the same meaning as in LTL_3, but $\top_p$ is possibly true and $\bot_p$ is possibly false.

RV-LTL Semantics

The semantics of RV-LTL is defined based on the semantics LTL_3 and FLTL:

$$[u \models_{RV} \varphi] = \begin{cases} \top & \text{if} \quad [u \models_3 \varphi] = \top \\ \bot & \text{if} \quad [u \models_3 \varphi] = \bot \\ \top_p & \text{if} \quad [u \models_3 \varphi] = ? \ \wedge \ [u \models_F \varphi] = \top \\ \bot_p & \text{if} \quad [u \models_3 \varphi] = ? \ \wedge \ [u \models_F \varphi] = \bot \end{cases}$$

RV-LTL

RV-LTL Monitor

Let φ be an LTL formula. The RV-LTL monitor of φ is the unique deterministic finite state machine $\mathcal{M}^{\varphi}_{RV} = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $\lambda : Q \to \mathbb{B}_4$, is a function such that:

$$\lambda(\delta(q_0, u)) = [u \models_{RV} \varphi]$$

for every finite word $u \in \Sigma^*$. □

RV-LTL Monitor

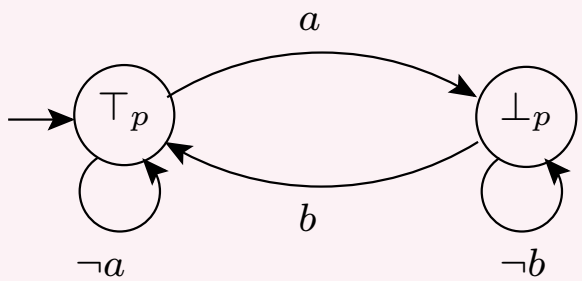
Let φ be an LTL formula. The RV-LTL monitor of φ is the unique deterministic finite state machine $\mathcal{M}^{\varphi}_{RV} = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $\lambda : Q \to \mathbb{B}_4$, is a function such that:

$$\lambda(\delta(q_0, u)) = [u \models_{RV} \varphi]$$

for every finite word $u \in \Sigma^*$. □

Example

RV-LTL monitor for $\mathbf{G}(a \to \mathbf{F}b)$



Presentation outline

Distributed Monitors

Distributed Monitors

Let $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ be a set of distributed monitors monitoring an underlying system.

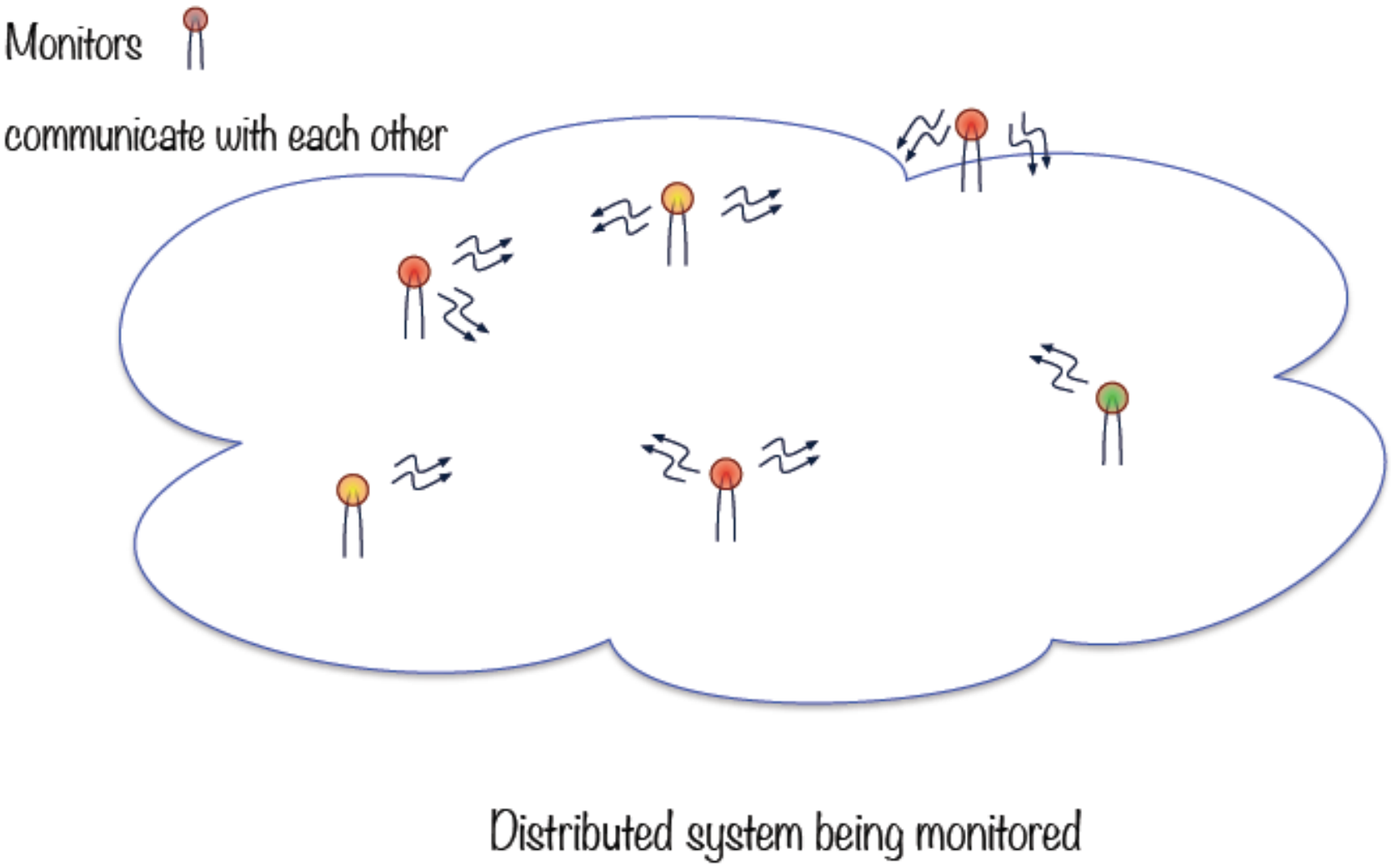
Distributed Monitors

Distributed Monitors

Let $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ be a set of distributed monitors monitoring an underlying system.

Each monitor $M_i \in \mathcal{M}$ takes a sample only once from the underlying system to obtain the values of propositions in AP as input.

Distributed Monitors (Not a nuclear power plant!)



Distributed Monitors

Local Snapshot

Each monitor M_i maintains an n registers, each of size $|AP|$ (i.e., $|AP| \times n$ local snapshot array LS^i, where

Distributed Monitors

Local Snapshot

Each monitor M_i maintains an n registers, each of size $|AP|$ (i.e., $|AP| \times n$ local snapshot array LS^i, where

- Register (i.e., column) i contains the partial view of monitor M_i (the sample taken by M_i);

Distributed Monitors

Local Snapshot

Each monitor M_i maintains an n registers, each of size $|AP|$ (i.e., $|AP| \times n$ local snapshot array LS^i, where

- Register (i.e., column) i contains the partial view of monitor M_i (the sample taken by M_i);
- each column $j \neq i$ ($1 \leq j \leq n$) contains M_i's local copy of monitor M_j's partial view (obtained through communication), and

Distributed Monitors

Local Snapshot

Each monitor M_i maintains an n registers, each of size $|AP|$ (i.e., $|AP| \times n$ local snapshot array LS^i, where

- Register (i.e., column) i contains the partial view of monitor M_i (the sample taken by M_i);
- each column $j \neq i$ ($1 \leq j \leq n$) contains M_i's local copy of monitor M_j's partial view (obtained through communication), and
- The value of each element in each local snapshot array ranges over $\{true, false, \natural\}$, where $\natural$ denotes an unknown value due to
  - partial of a monitor,
  - a monitor crash; or
  - communication delays.

  All elements of all local snapshot arrays are initialized to $\natural$.

Distributed Monitors

Distributed Monitors

Shared Memory

Monitors communicate through a shared memory array SM of size $|AP| \times n$.

Distributed Monitors

Shared Memory

Monitors communicate through a shared memory array SM of size $|AP| \times n$.

Monitor Communication

Each monitor M_i can perform one of the following actions:

- A write action by monitor M_i writes the content of LS_i^i into SM.
- A snapshot action by monitor M_i writes the entire content of SM into LS^i.

Distributed Monitors

Shared Memory

Monitors communicate through a shared memory array SM of size $|AP| \times n$.

Monitor Communication

Each monitor M_i can perform one of the following actions:

- A write action by monitor M_i writes the content of LS^i_i into SM.
- A snapshot action by monitor M_i writes the entire content of SM into LS^i.

Monitor Behavior

```
Monitor(){
    take_sample();
    repeat
        write();
        snapshot();
    until(...)
    emit [L̂S^i ⊨ φ]; //L̂S^i is the sequence local snapshots in M_i.
```

Distributed Monitors

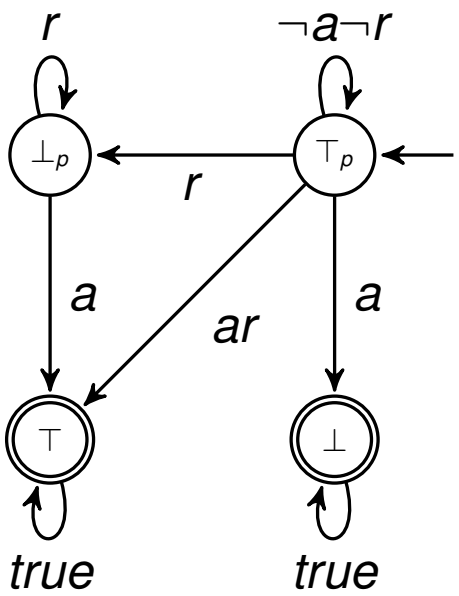
Example

Consider the following request/acknowledgment property:

- if a request is emitted (i.e., $r = true$), then it should eventually be acknowledged (i.e., $a = true$)
- an acknowledgment happens only in response to a request.

$$\varphi_{ra_1} = \mathbf{G}(\neg a \neg r) \vee [(\neg a \, \mathbf{U} \, r) \wedge \mathbf{F} a]$$

RV-LTL Monitor



Distributed Monitors

Example

$$\varphi_{ra_2} = \{\mathbf{G}(\neg a_1 \neg r_1) \vee [(\neg a_1 \,\mathbf{U}\, r_1) \wedge \mathbf{F} a_1]\} \wedge \{\mathbf{G}(\neg a_2 \neg r_2) \vee [(\neg a_2 \,\mathbf{U}\, r_2) \wedge \mathbf{F} a_2]\}$$

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

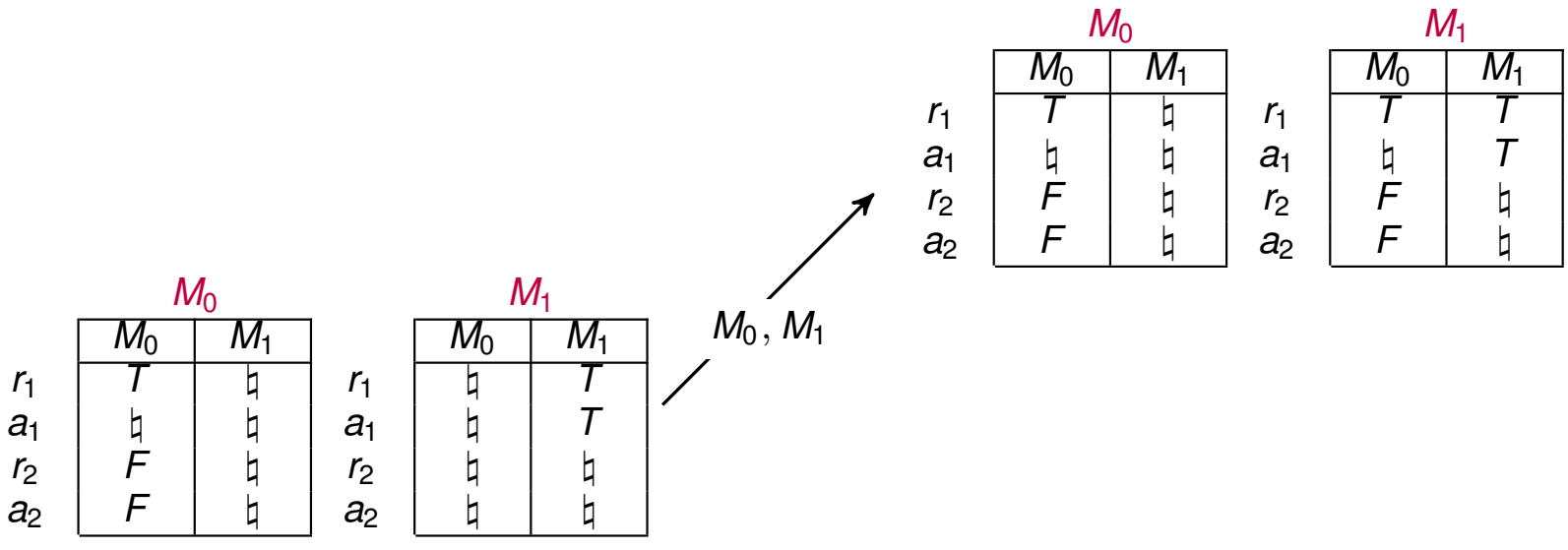
M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

Distributed Monitors

Example

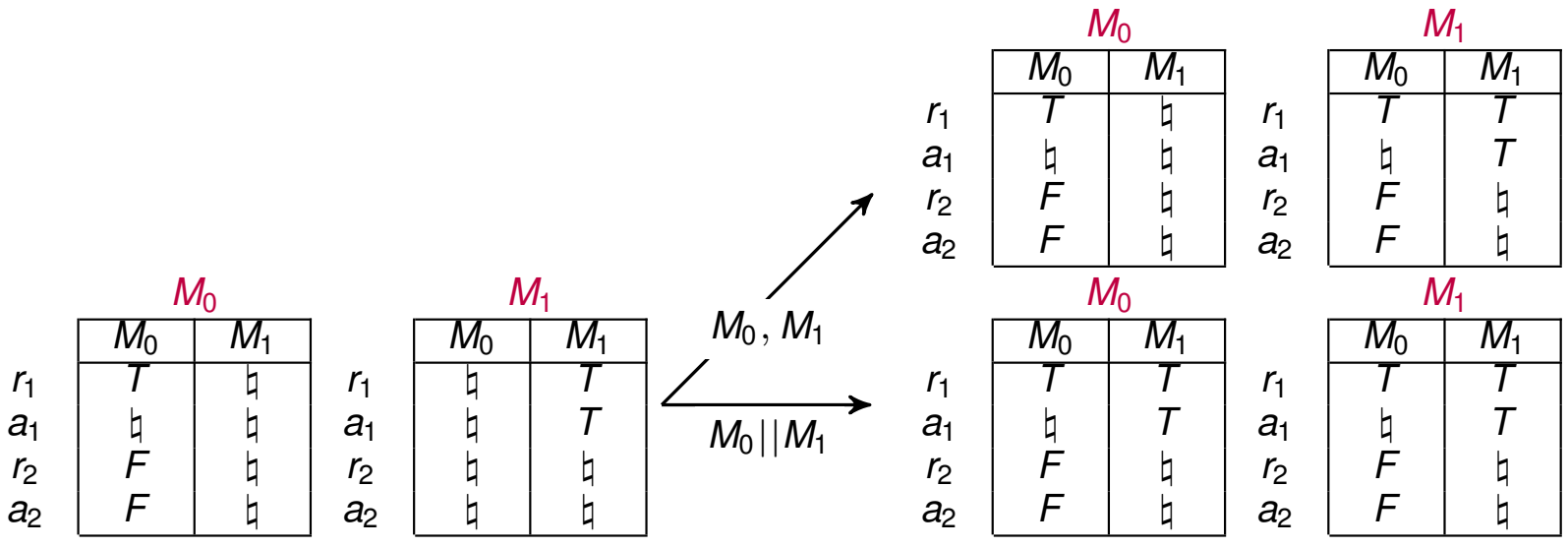
$$\varphi_{ra_2} = \{\mathbf{G}(\neg a_1 \neg r_1) \vee [(\neg a_1 \,\mathbf{U}\, r_1) \wedge \mathbf{F} a_1]\} \wedge \{\mathbf{G}(\neg a_2 \neg r_2) \vee [(\neg a_2 \,\mathbf{U}\, r_2) \wedge \mathbf{F} a_2]\}$$



Distributed Monitors

Example

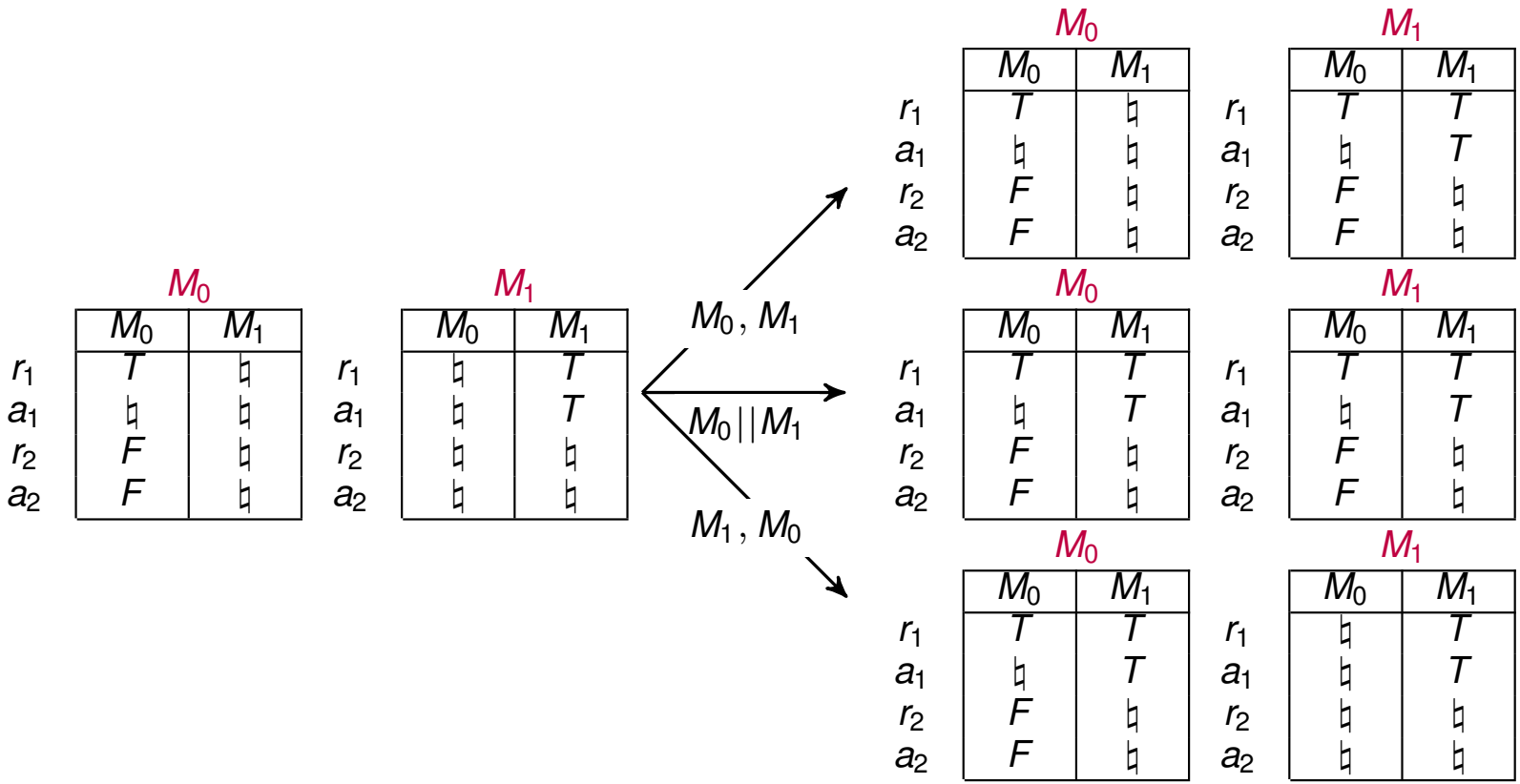
$$\varphi_{ra_2} = \{\mathbf{G}(\neg a_1 \neg r_1) \vee [(\neg a_1 \,\mathbf{U}\, r_1) \wedge \mathbf{F} a_1]\} \wedge \{\mathbf{G}(\neg a_2 \neg r_2) \vee [(\neg a_2 \,\mathbf{U}\, r_2) \wedge \mathbf{F} a_2]\}$$



Distributed Monitors

Example

$$\varphi_{ra_2} = \{\mathbf{G}(\neg a_1 \neg r_1) \vee [(\neg a_1 \,\mathbf{U}\, r_1) \wedge \mathbf{F} a_1]\} \wedge \{\mathbf{G}(\neg a_2 \neg r_2) \vee [(\neg a_2 \,\mathbf{U}\, r_2) \wedge \mathbf{F} a_2]\}$$



Distributed Monitors

Assumption

Monitors do not read inconsistent samples.

Distributed Monitors

Assumption

Monitors do not read inconsistent samples.

Local Formula Evaluation

Distributed Monitors

Assumption

Monitors do not read inconsistent samples.

Local Formula Evaluation

For each atomic proposition $ap \in AP$, all monitors are provided with an n-ary function

$$\mathcal{F}_{ap} : \{true, false, \natural\}^n \rightarrow \{true, false\}$$

where n is the number of monitors.

Distributed Monitors

Assumption

Monitors do not read inconsistent samples.

Local Formula Evaluation

For each atomic proposition $ap \in AP$, all monitors are provided with an n-ary function

$$\mathcal{F}_{ap} : \{true, false, \natural\}^n \rightarrow \{true, false\}$$

where n is the number of monitors.

Example

For all atomic propositions a_1, r_1, a_2, r_2, given two values v_1 and v_2, we have

$$\mathcal{F}_*(v_1, v_2) = \begin{cases} true & \text{if } (v_1 = true) \vee (v_2 = true) \\ false & \text{otherwise} \end{cases}$$

Distributed Monitors

Example

M_0

	M_0	M_1	
r_1	T	$\natural$	$\mathcal{F}(T, \natural) = T$
a_1	$\natural$	$\natural$	$\mathcal{F}(F, \natural) = F$
r_2	F	$\natural$	$\mathcal{F}(F, \natural) = F$
a_2	F	$\natural$	$\mathcal{F}(F, \natural) = F$

$$[\mathcal{F}(LS^0) \models_{RV} \varphi_{ra_2}] = \bot_p$$

M_1

	M_0	M_1	
r_1	$\natural$	T	$\mathcal{F}(\natural, T) = T$
a_1	$\natural$	T	$\mathcal{F}(\natural, T) = T$
r_2	$\natural$	$\natural$	$\mathcal{F}(\natural, F) = F$
a_2	$\natural$	$\natural$	$\mathcal{F}(\natural, F) = F$

$$[\mathcal{F}(LS^1) \models_{RV} \varphi_{ra_2}] = \top_p$$

In the underlying system: $[u \models_{RV} \varphi] = \top_p$

Distributed Monitors

Global Consistency

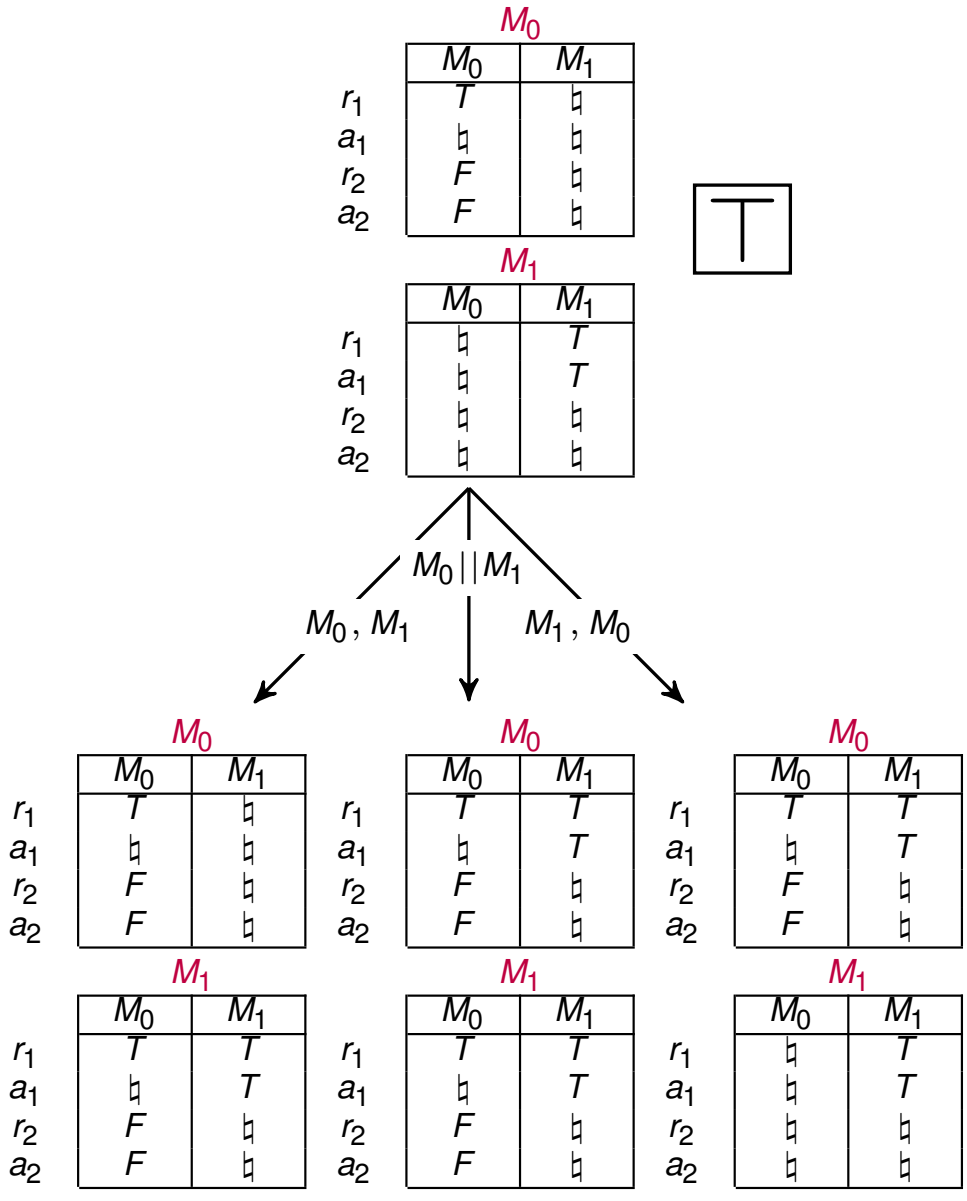
Let u and u' be two finite words where

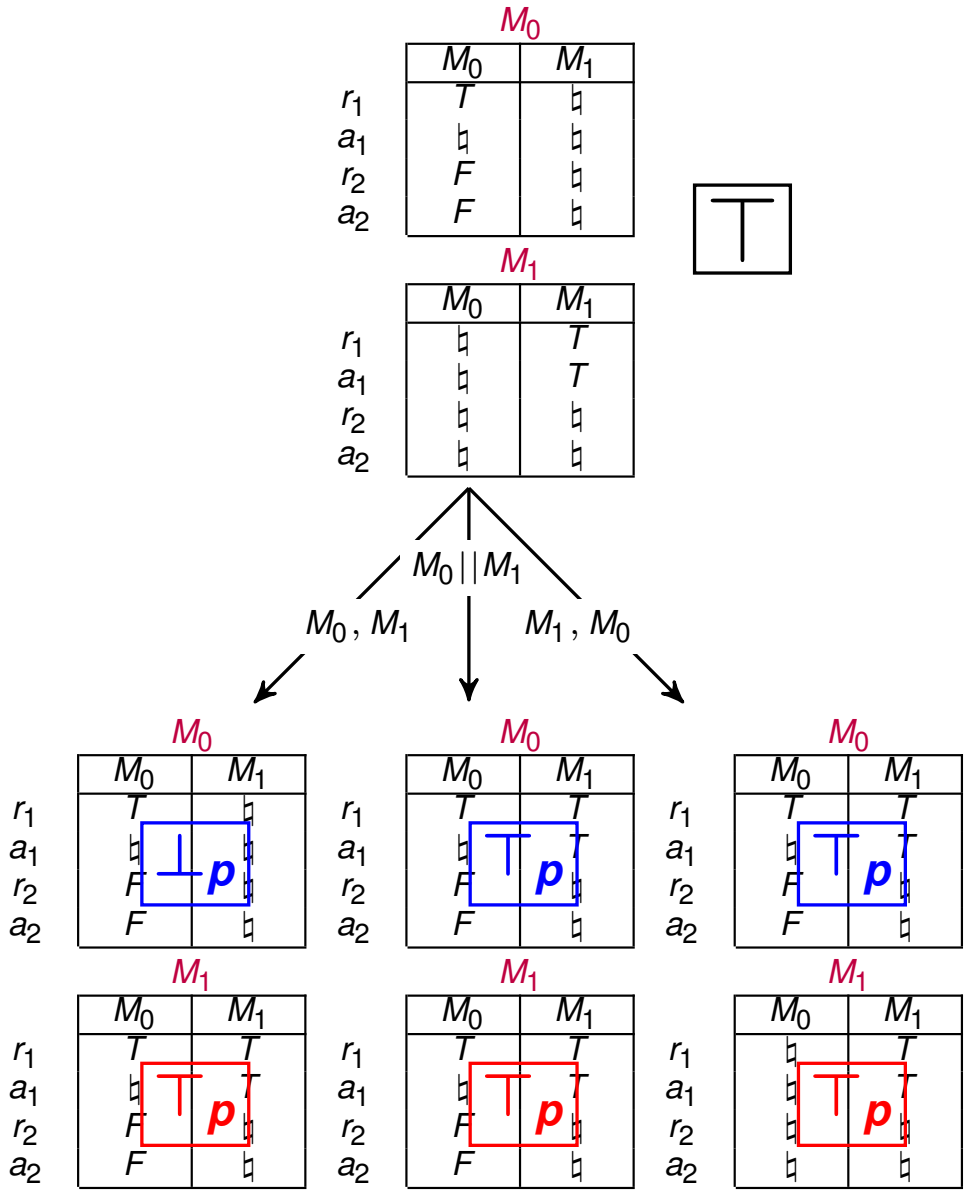
$$[u \models_F \varphi] = \bot$$

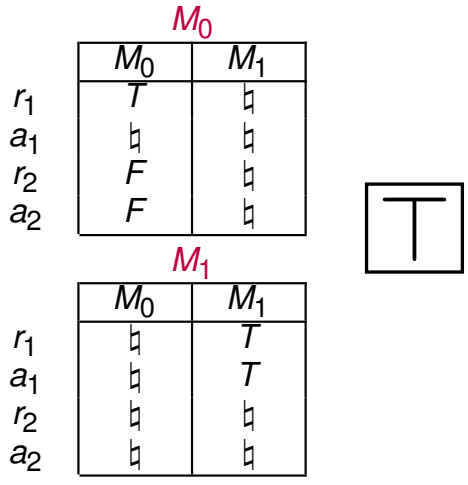
and

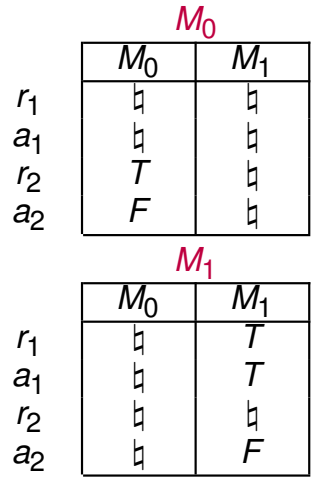
$$[u' \models_F \varphi] = \top$$

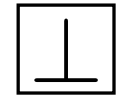
We say that a set $\mathcal{M}$ of monitors respect global consistency iff the set of verdicts emitted by monitors in $\mathcal{M}$ for u in any communication interleaving is different from the set of verdicts emitted by monitors in $\mathcal{M}$ for u' in any communication interleaving.

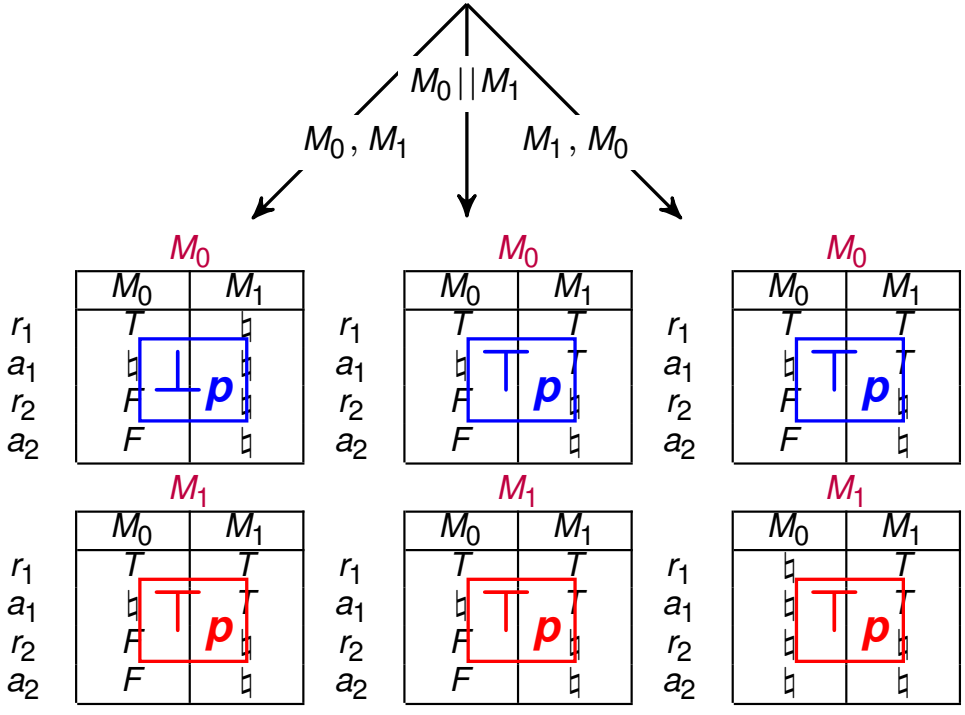

M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯
$M_0 \,||\, M_1$
M_0 , M_1
M_1 , M_0
M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_0
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_0
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯

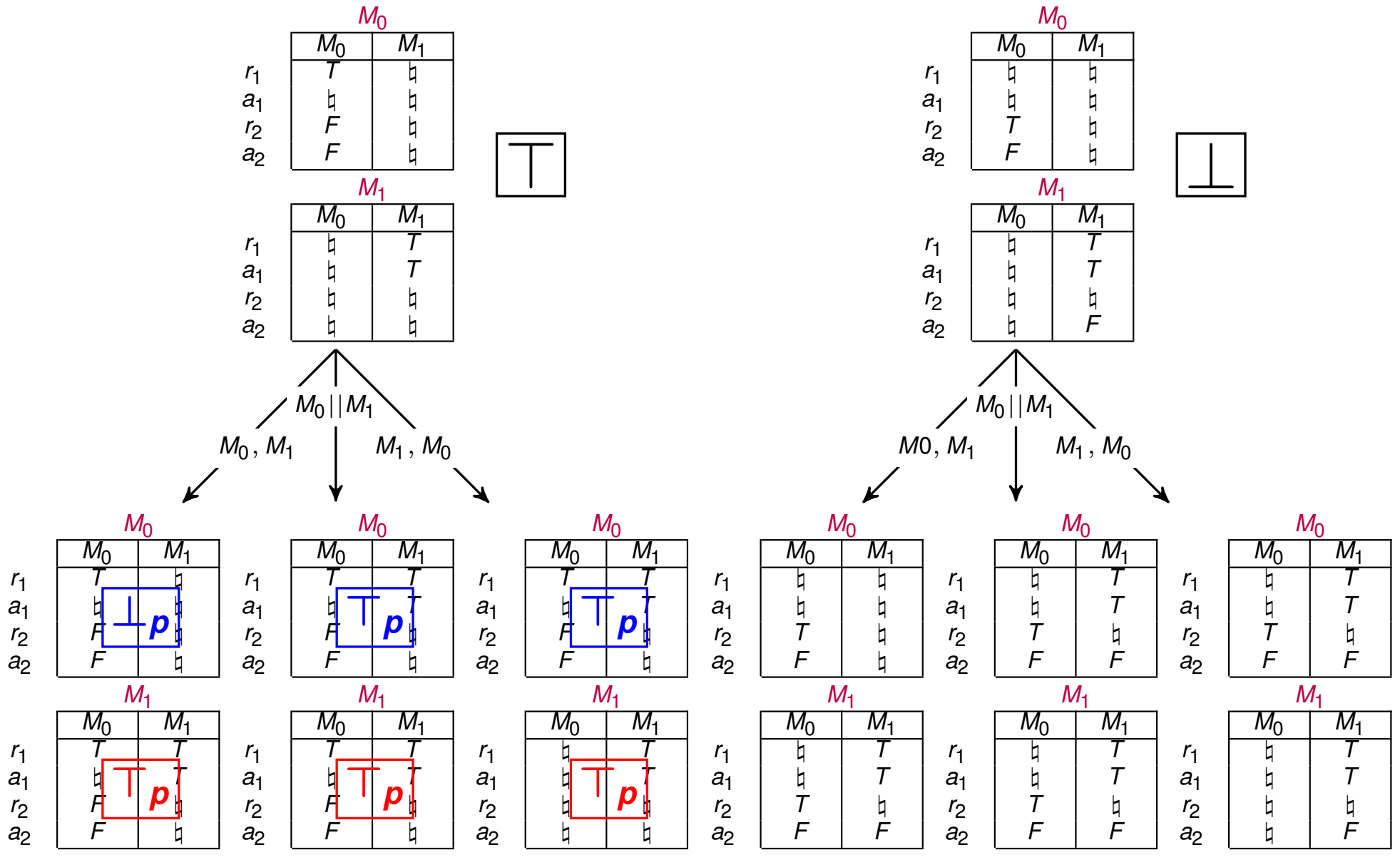

M_0
M_0 M_1
r_1 T ␣
a_1 ␣ ␣
r_2 F ␣
a_2 F ␣
M_1
M_0 M_1
r_1 ␣ T
a_1 ␣ T
r_2 ␣ ␣
a_2 ␣ ␣
$M_0 \,||\, M_1$
M_0 , M_1
M_1 , M_0
p

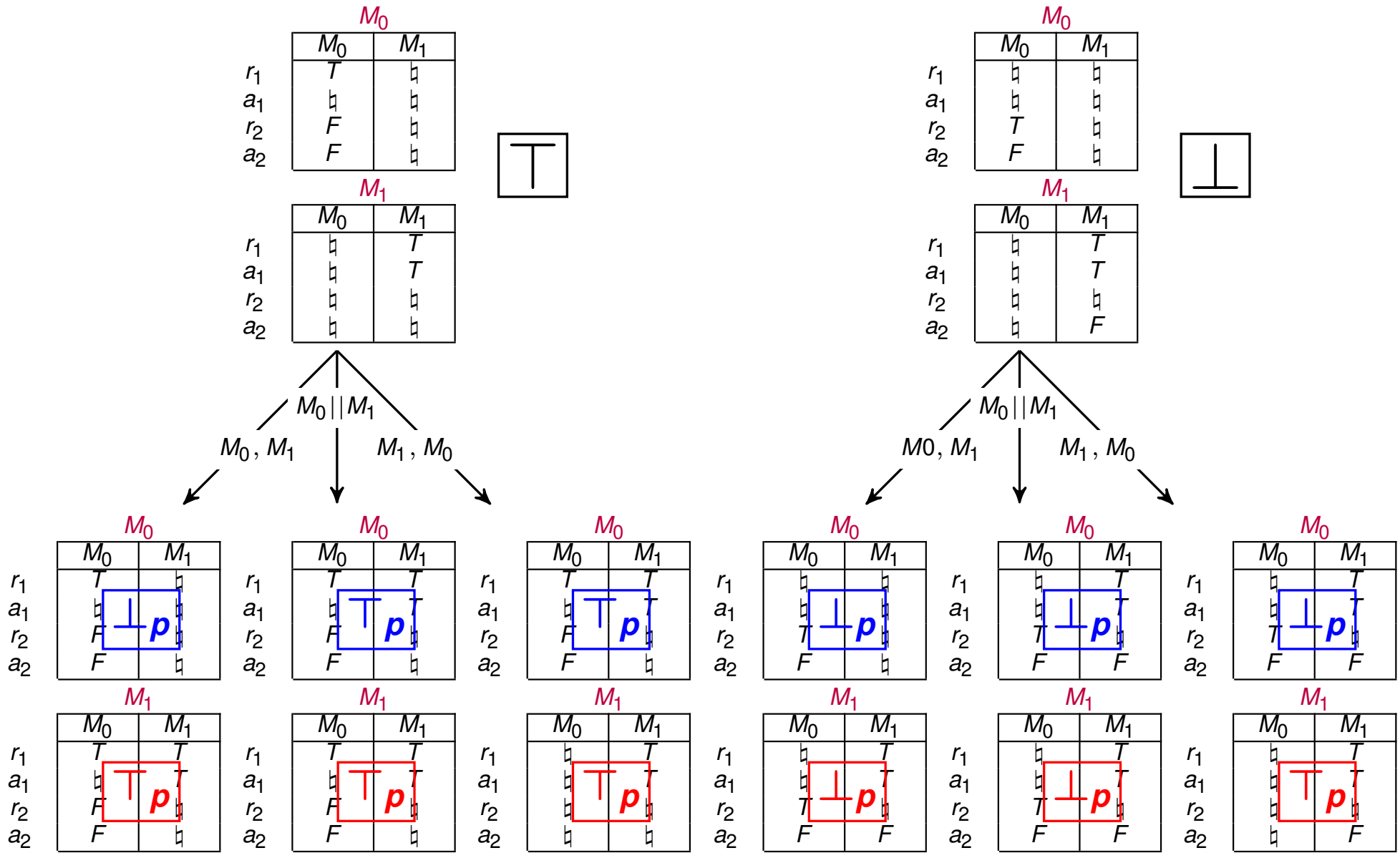

M_0
M_0 M_1
r_1 T ␣
a_1 ␣ ␣
r_2 F ␣
a_2 F ␣
M_1
M_0 M_1
r_1 ␣ T
a_1 ␣ T
r_2 ␣ ␣
a_2 ␣ ␣

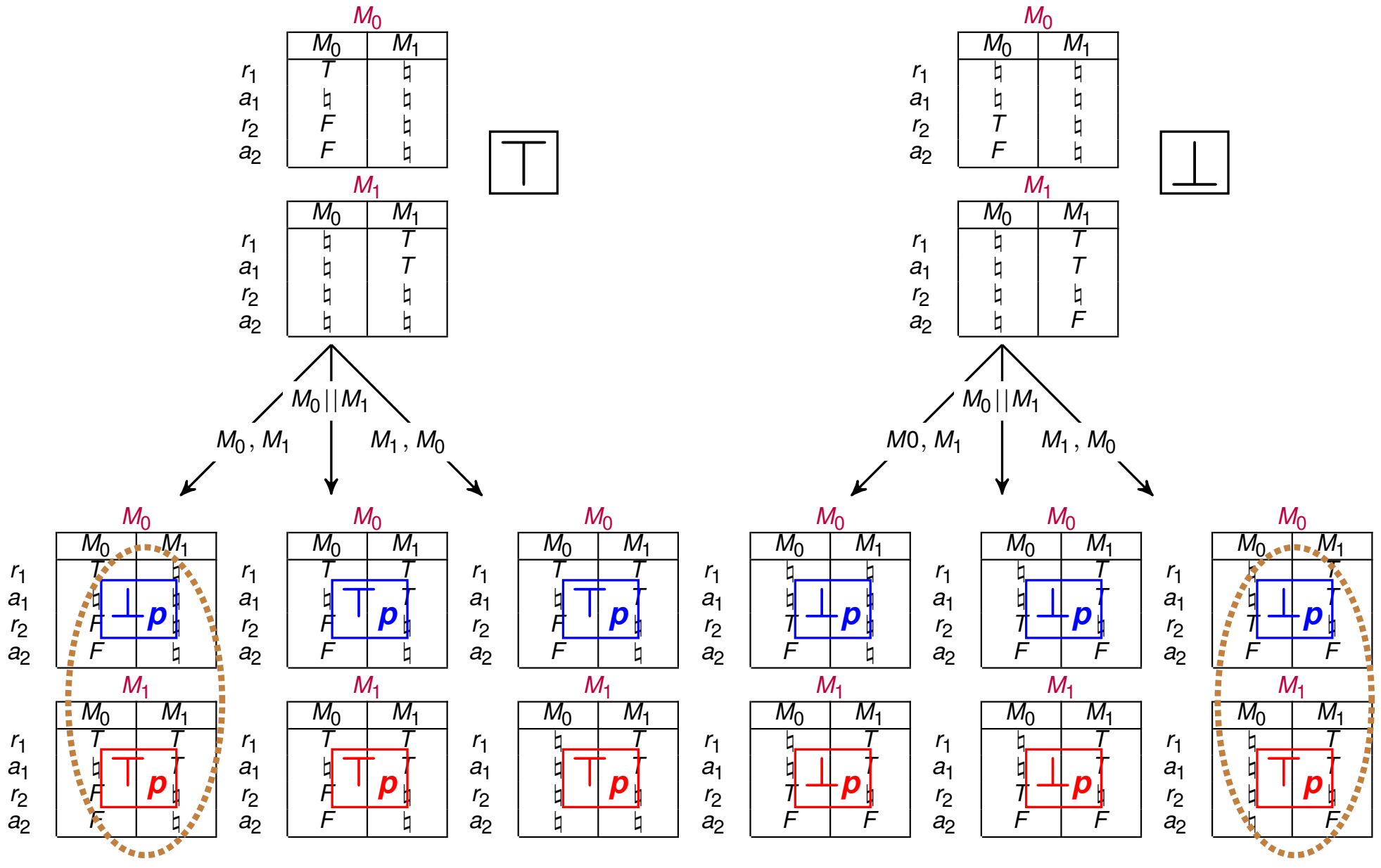

M_0
M_0 M_1
r_1 ␣ ␣
a_1 ␣ ␣
r_2 T ␣
a_2 F ␣
M_1
M_0 M_1
r_1 ␣ T
a_1 ␣ T
r_2 ␣ ␣
a_2 ␣ F

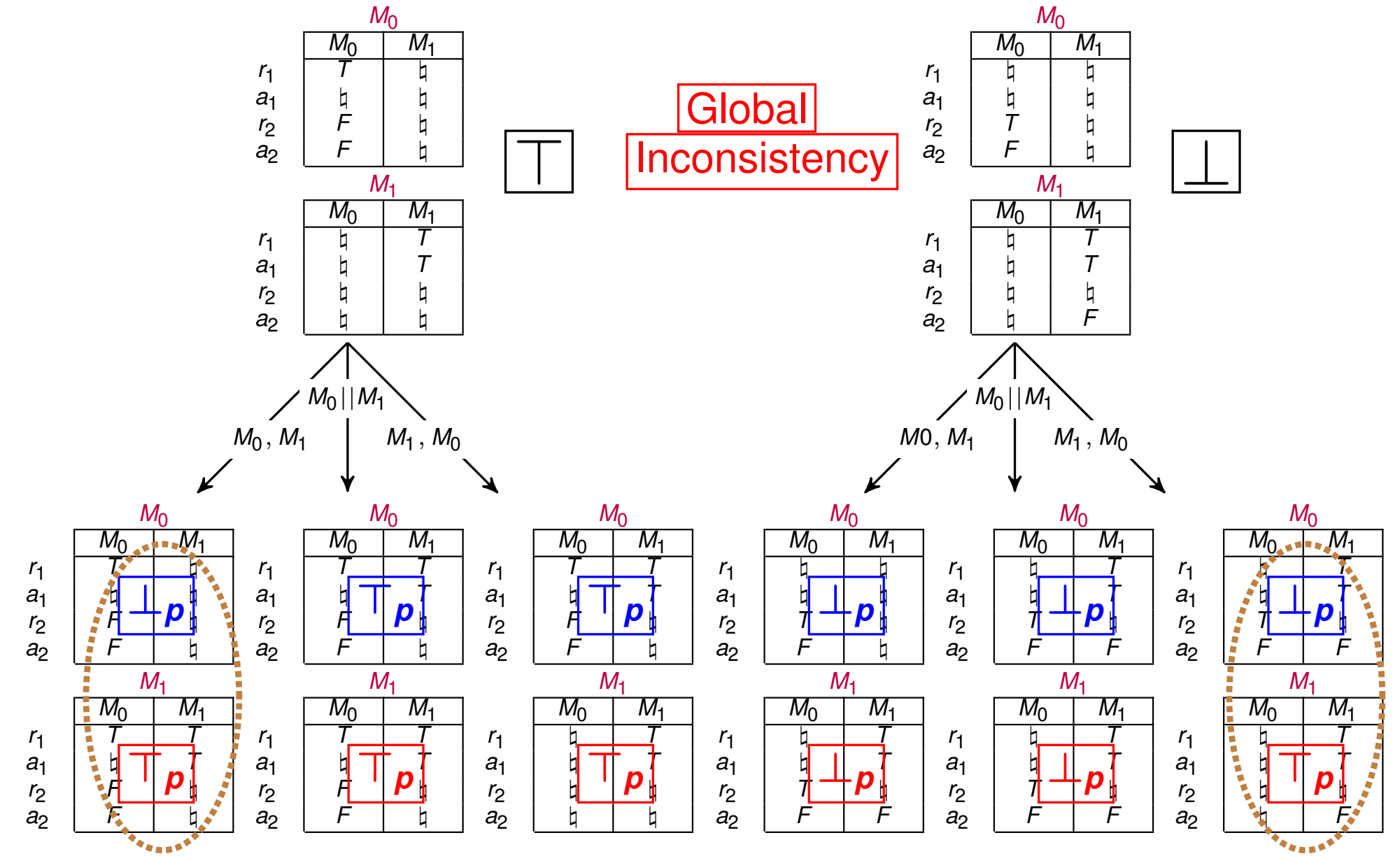



$M_0 \,||\, M_1$
M_0 , M_1
M_1 , M_0
M_0
M_0 M_1
r_1 T ␣
a_1 ␣ ␣
r_2 F ␣
a_2 F ␣
p
M_1
M_0 M_1
r_1 T T
a_1 ␣ T
r_2 F ␣
a_2 F ␣
p


M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0, M_1
M_1, M_0
M0, M1
p


M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0, M_1
M_1, M_0
M0, M1
p


M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0, M_1
M_1, M_0
M0, M_1
p


Global
Inconsistency
M_0
M_1
$M_0 || M_1$
M_0, M_1
M_1, M_0
M0, M_1
r_1
a_1
r_2
a_2
p

General Lower bound Results

Lemma

Not all LTL formulas can be consistently monitored by a 1-round distributed monitor with traces in RV-LTL, even if monitors satisfy state coverage, and even if no monitors crash during the execution of the monitor.

theorem

Not all LTL formulas can be consistently monitored by a distributed monitor with traces in RV-LTL, even if monitors satisfy state coverage, even if no monitors crash during the execution of the monitor, and even if the monitors perform an arbitrarily large number of rounds.

Presentation outline

Alternation Number

Idea

In a word, we count the number of times that the valuation of a formula may change from.

Alternation Number

Idea

In a word, we count the number of times that the valuation of a formula may change from.

Alternation number

The alternation number of an LTL formula φ is the following:

$$AN(\varphi) = \max\left\{A(w) \mid w \in \Sigma^*\right\}$$

where

$$A(w) = \begin{cases} A(w') + 1 & \text{if} \quad [w \models_F \varphi] \neq [w' \models_F \varphi] \\ 0 & \text{if} \quad length(w) = 1 \end{cases}$$

where w' denotes the longest proper prefix of w. □

Obtaining Alternation Number

Theorem

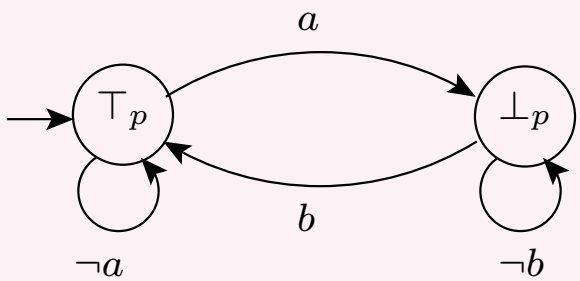
The alternation number of LTL formula φ is the length of the longest walk of the RV-LTL monitor of φ.

Obtaining Alternation Number

Theorem

The alternation number of LTL formula φ is the length of the longest walk of the RV-LTL monitor of φ.

Example



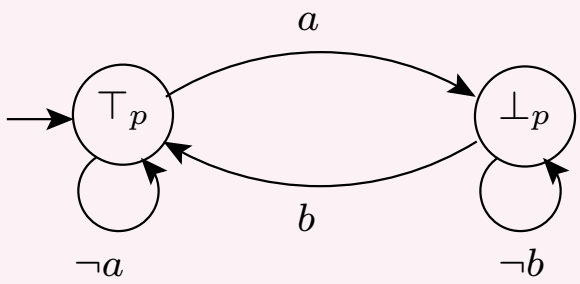
$AN(\mathbf{G}(a \rightarrow \mathbf{F}b)) = \infty$

Obtaining Alternation Number

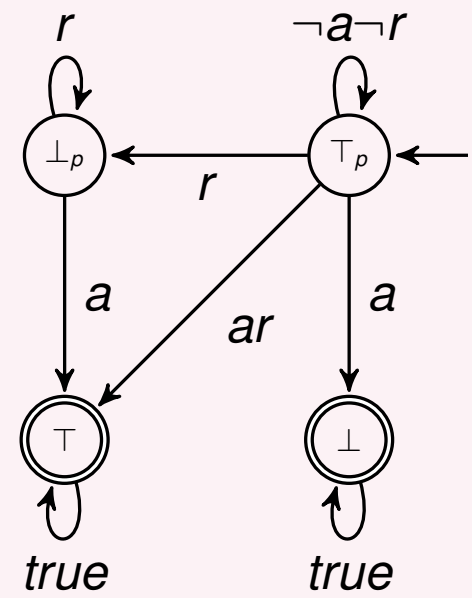
Theorem

The alternation number of LTL formula φ is the length of the longest walk of the RV-LTL monitor of φ.

Example



$AN(\mathbf{G}(a \rightarrow \mathbf{F}b)) = \infty$



$AN(\mathbf{G}(\neg a \neg r) \vee [(\neg a \mathbf{U} r) \wedge \mathbf{F}a]) = 2$

Global Consistency

Theorem

In order to monitor an LTL formula φ by a wait-free distributed monitor, we need at least $AN(\varphi) + 1$ truth values to ensure global consistency.

LTL_{2k+4}

Truth Values

LTL_{2k+4} has $2K + 4$ truth values: $\mathbb{B}_K = \{\bot_0, \top_0, \bot_1, \top_1, \cdots \bot_k, \top_k, \bot, \top\}$

LTL_{2k+4}

Truth Values

LTL_{2k+4} has $2K+4$ truth values: $\mathbb{B}_K = \{\bot_0, \top_0, \bot_1, \top_1, \cdots \bot_k, \top_k, \bot, \top\}$

Semantics

$$
[u \models_{2k+4} \varphi] = \begin{cases}
\bot & \text{if} \quad [u \models_3 \varphi] = \bot \\
\top & \text{if} \quad [u \models_3 \varphi] = \top \\
\bot_0 & \text{if} \quad |u| = 1 \ \wedge \ [u \models_3 \varphi] = ? \ \wedge \ [u \models_F \varphi] = \bot \\
\top_0 & \text{if} \quad |u| = 1 \ \wedge \ [u \models_3 \varphi] = ? \ \wedge \ [u \models_F \varphi] = \top \\
\top_i \quad \text{with } i \in [0,k] & \text{if} \quad |u| \geq 2 \ \wedge \ [u \models_3 \varphi] = ? \ \wedge \ [u \models_F \varphi] = \top \ \wedge \\
 & \quad [u' \models_{2k+4} \varphi] \in \{\top_i, \bot_i\} \\
\bot_i \quad \text{with } i \in [0,k) & \text{if} \quad (|u| \geq 2 \ \wedge \ [u \models_3 \varphi] = ? \ \wedge \ [u \models_F \varphi] = \bot) \ \wedge \\
 & \quad ([u' \models_{2k+4} \varphi] = \bot_i \ \vee \ [u' \models_{2k+4} \varphi] = \top_{i-1}) \\
\bot_k & \text{if} \quad (|u| \geq 2 \ \wedge \ [u \models_3 \varphi] = ? \ \wedge \ [u \models_F \varphi] = \bot) \ \wedge \\
 & \quad ([u' \models_{2k+4} \varphi] = \bot_k \ \vee \ [u' \models_{2k+4} \varphi] = \top_k \ \vee \\
 & \quad\quad [u' \models_{2k+4} \varphi] = \top_{k-1})
\end{cases}
$$

LTL_{2k+4} Monitor Construction

LTL_{2k+4} Monitor

Let φ be an LTL formula. The LTL_{2k+4} monitor of φ is the unique deterministic finite state machine $\mathcal{M}_k^\varphi = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is a set of states, q_0 is the initial state, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and λ is a function that maps each state in Q to a value in $\mathbb{B}_K$, such that:

$$[u \models_K \varphi] = \lambda(\delta(q_0, u)),$$

for every finite word $u \in \Sigma^*$.

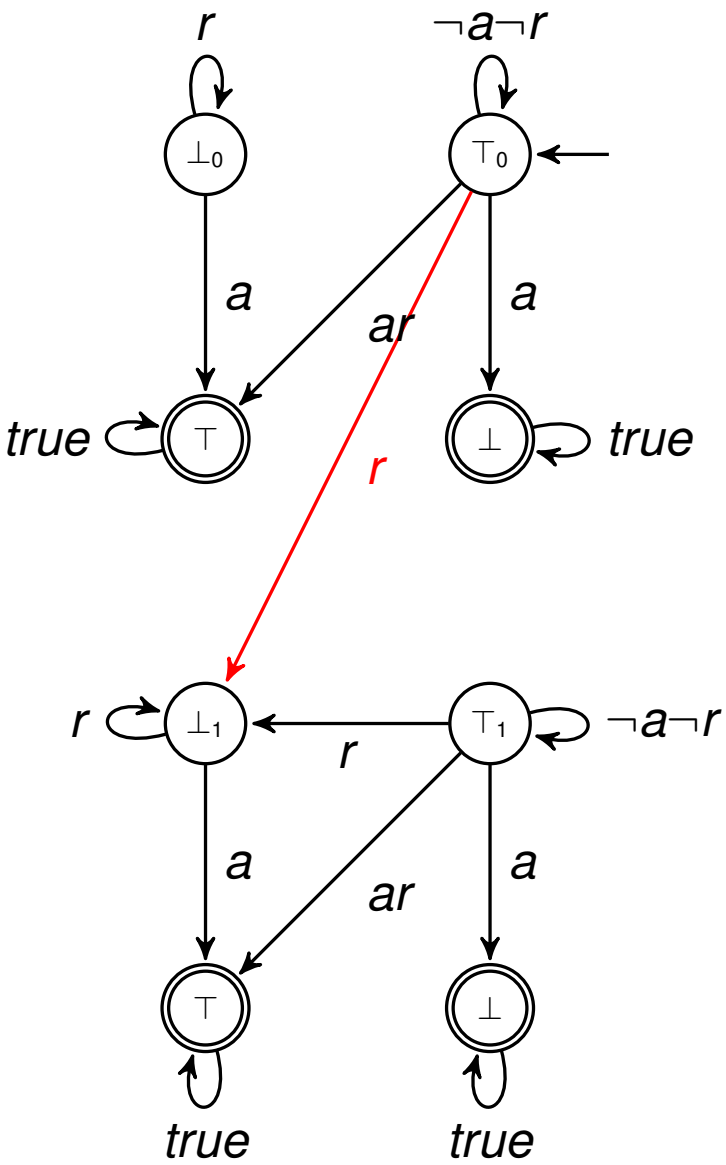
LTL_{2k+4} Monitor Construction Algorithm

Input: Alphabet Σ, LTL formula φ, $K \in \mathbb{Z}_{\geq 0}$
Output: LTL_{2k+4} monitor
$M_k^{\varphi} = (\Sigma, Q, q_0, \delta, \lambda)$

1 $(Q, q_0, \delta, \lambda) \leftarrow$ ConstructMonitor(Σ, φ, 0);
2 **for** $k \leftarrow 1$ **to** K **do**
3 $\quad (\bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda}) \leftarrow$ ConstructMonitor(Σ, φ, k);
4 $\quad Q \leftarrow Q \cup \bar{Q}$; $\delta \leftarrow \delta \cup \bar{\delta}$; $\lambda \leftarrow \lambda \cup \bar{\lambda}$;
5 $\quad$ **forall the** $q \in Q$, $\bar{q} \in \bar{Q}$ **do**
6 $\quad\quad$ **if** $(\lambda(q) = \top_{k-1} \wedge \lambda(\bar{q}) = \bot_k \wedge k - 1 \leq K)$ **then**
7 $\quad\quad\quad$ **forall the** $q' \in Q$, $a \in \Sigma$ **do**
8 $\quad\quad\quad\quad$ **if** $\lambda(q') = \bot_{k-1} \wedge \delta(q, a) = q'$ **then**
9 $\quad\quad\quad\quad\quad \delta = \delta - \{(q, a, q')\}$;
10 $\quad\quad\quad\quad\quad \delta = \delta \cup \{(q, a, \bar{q})\}$;
11 **return** $M_k^{\varphi} = (\Sigma, Q, q_0, \delta, \lambda)$;

1 **Function** `ConstructMonitor(`*alphabet* Σ, LTL *formula* φ, **int** k`)`
2 **Let** $\mathcal{M}_3^{\varphi} = (\Sigma, Q, q_0, \delta, \lambda)$ and $\mathcal{M}_F^{\varphi} = (\Sigma, Q', q_0', \delta', \lambda')$;
3 $\bar{Q} \leftarrow Q \times Q'$;
4 $\bar{q}_0 \leftarrow (q_0, q_0')$;
5 **forall the** $q \in Q$, $q' \in Q'$ **do**
6 $\quad \bar{\delta}((q, q'), a) = (\delta(q, a), \delta'(q', a))$;
7 $\quad$ **if** $(\lambda(q) \neq ?)$ **then**
8 $\quad\quad \bar{\lambda}((q, q')) \leftarrow \lambda(q)$;
9 $\quad$ **else**
10 $\quad\quad$ **if** $(\lambda(q) = ? \wedge \lambda'(q') = \top)$ **then**
11 $\quad\quad\quad \bar{\lambda}((q, q')) \leftarrow \top_k$;
12 $\quad\quad$ **else**
13 $\quad\quad\quad$ **if** $(\lambda(q) = ? \wedge \lambda'(q') = \bot)$ **then**
14 $\quad\quad\quad\quad \bar{\lambda}((q, q')) \leftarrow \bot_k$;
15 **return** $(\bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$;

LTL_{2k+4} Monitor Construction



Monitor for

$\Box(\neg a \neg r) \vee [(\neg a \,\mathbf{U}\, r) \wedge \Diamond a]$

in LTL_6.

LTL_{2k+4} Verdict Inference

Effect of Interleavings

When a local monitor takes a snapshot, it advances its monitor state based on the highest possible level of interleavings that may lead to this snapshot.

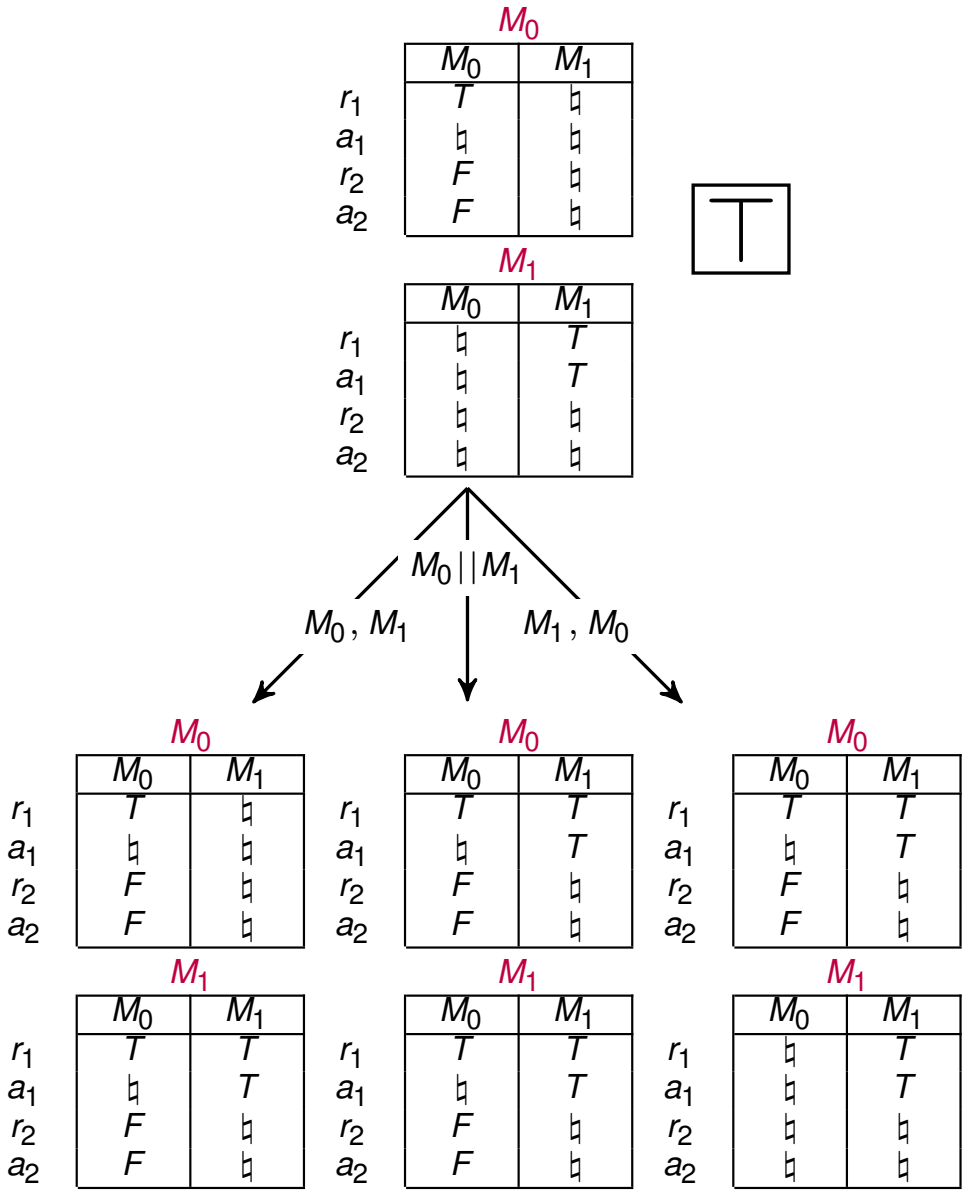
Example

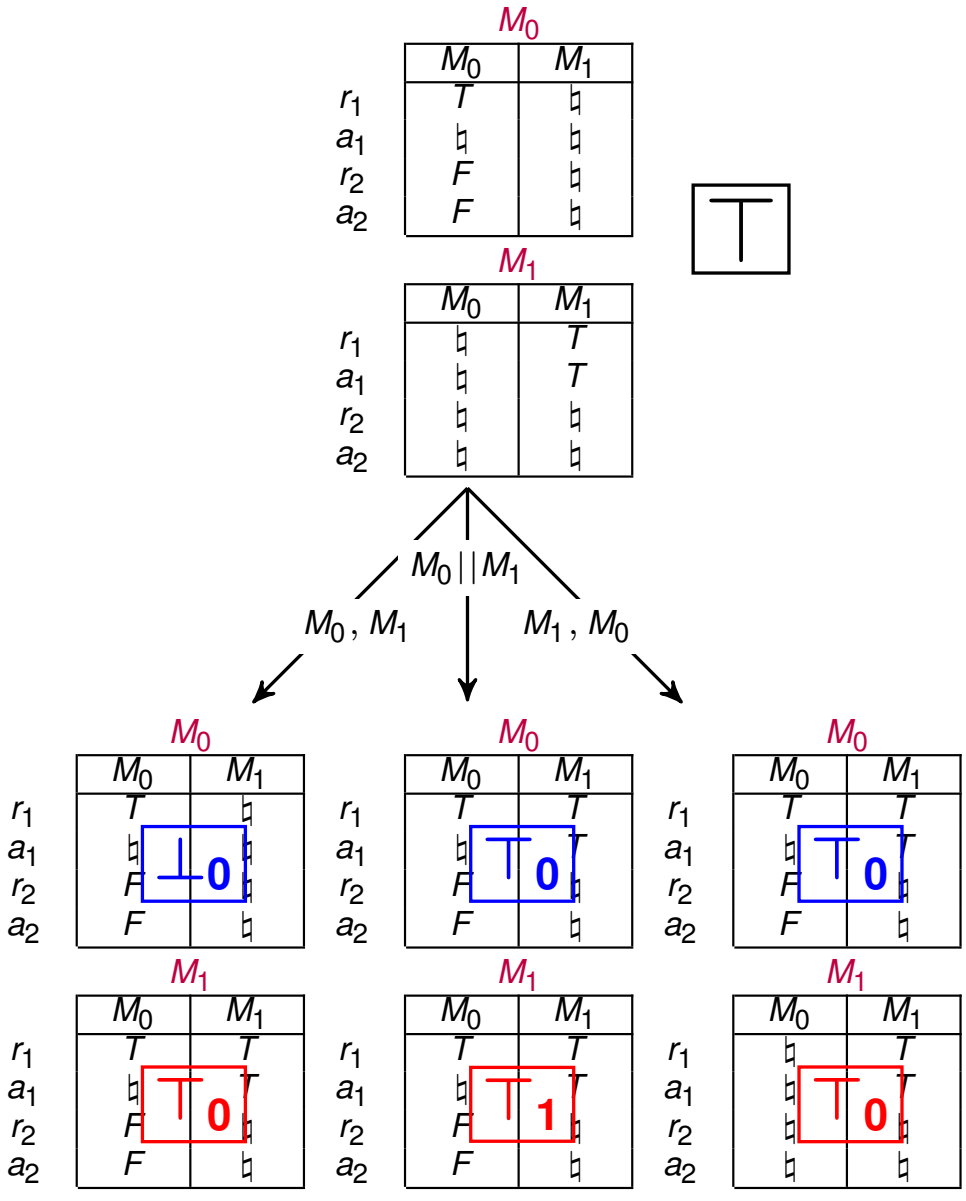
In our request/acknowledgment property, global state $s = \{r_1, a_1, r_2, a_2\}$ can be reached by either word

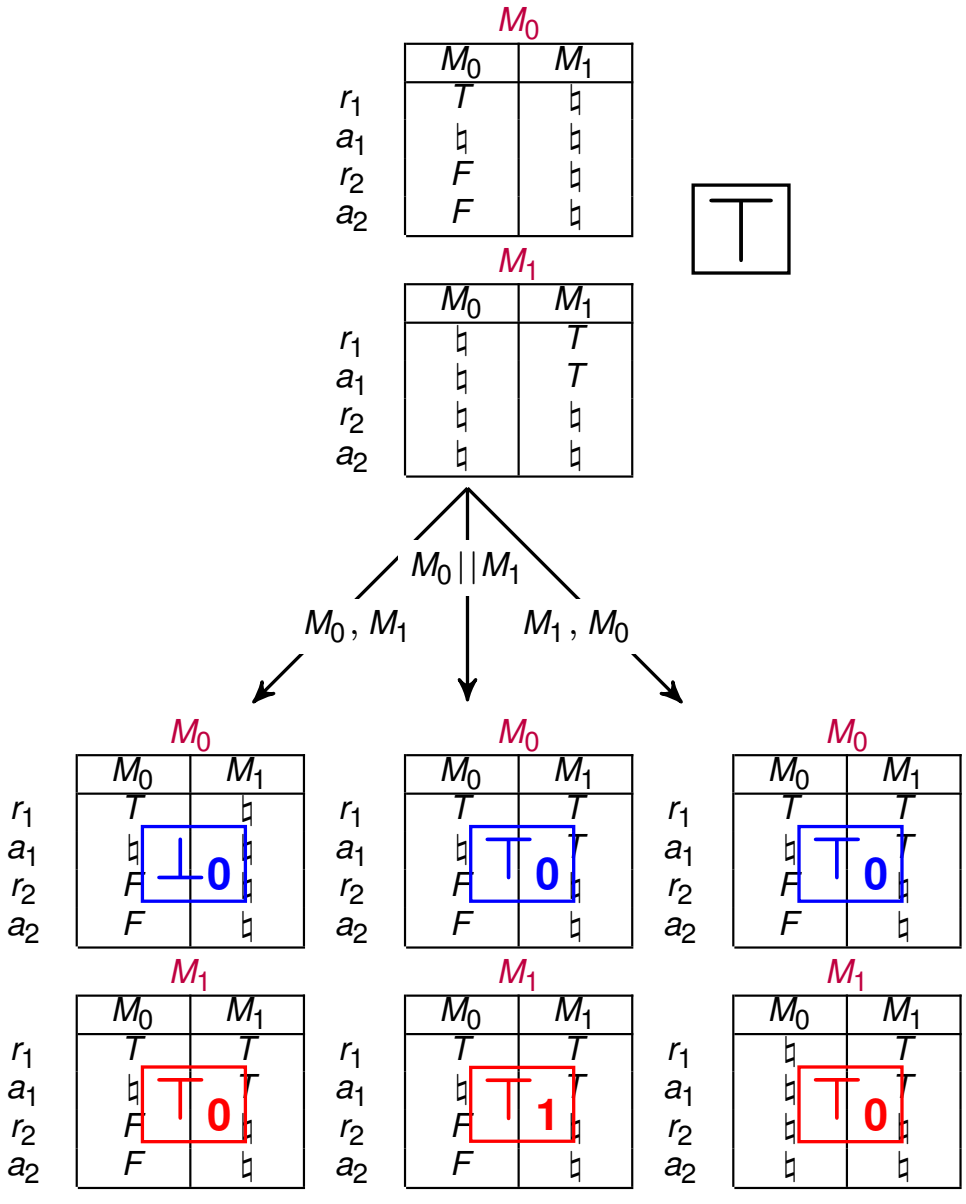
- $w_1 = \{r_1\}\{r_1, a_1\}\{r_1, a_1, r_2\}\{r_1, a_1, r_2, a_2\}$
- $w_2 = \{r_1\}\{r_1, r_2\}\{r_1, a_1, r_2\}\{r_1, a_1, r_2, a_2\}$.

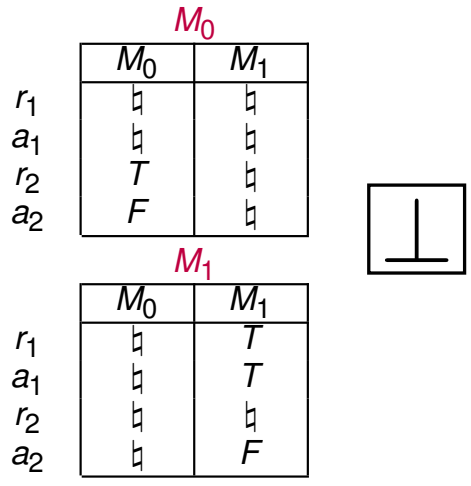
Evaluating s

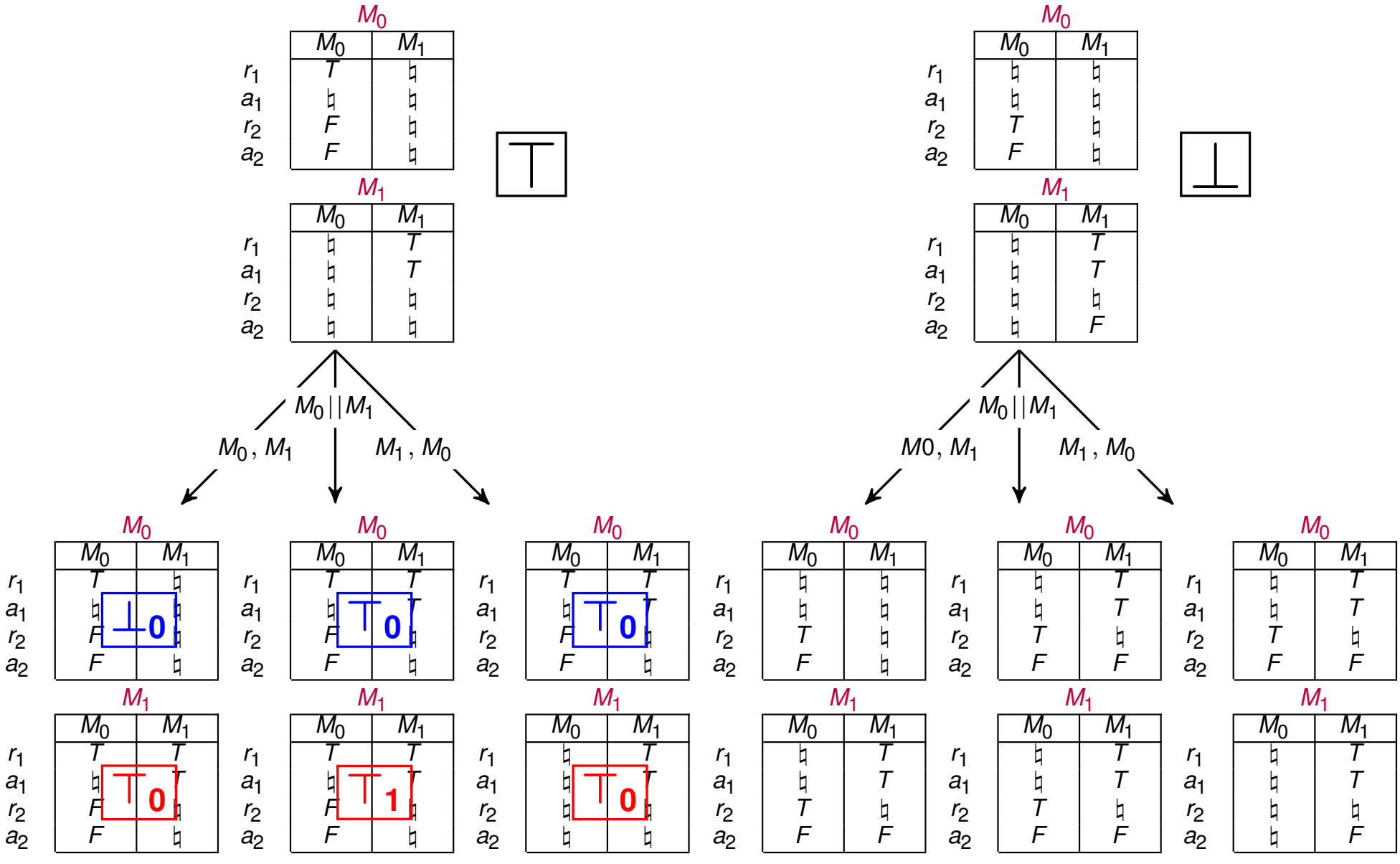
- through w_1 results in $\top_1$
- through w_2 results in $\top_0$.

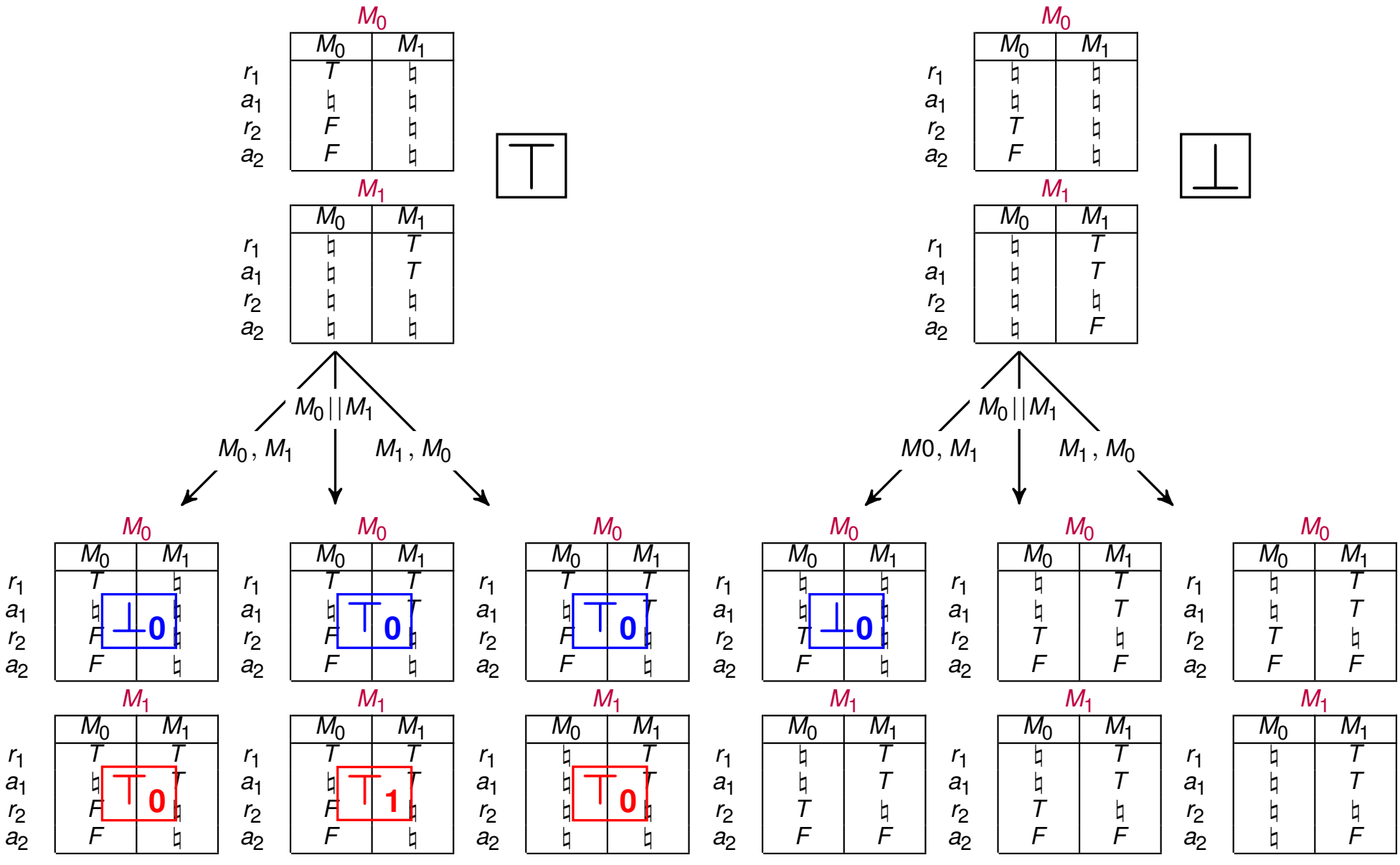

M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_0
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_0
M_0 M_1
r_1 T T
a_1 ↯ T
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯

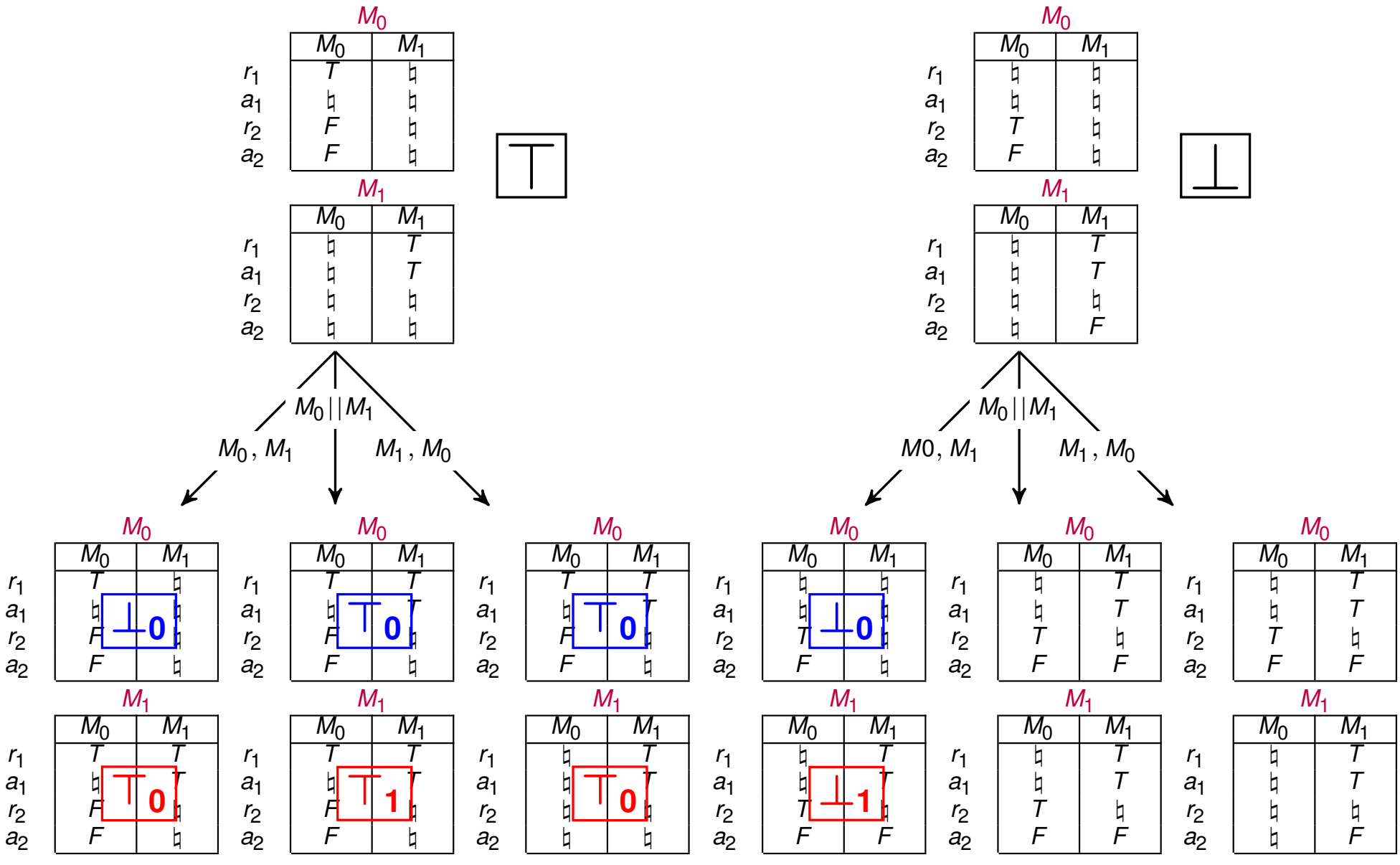

M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ⊥ 0
r_2 F ↯
a_2 F ↯
M_0
M_0 M_1
r_1 T T
a_1 ↯ ⊤ 0
r_2 F ↯
a_2 F ↯
M_0
M_0 M_1
r_1 T T
a_1 ↯ ⊤ 0
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ ⊤ 0
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ ⊤ 1
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ ⊤ 0
r_2 ↯ ↯
a_2 ↯ ↯

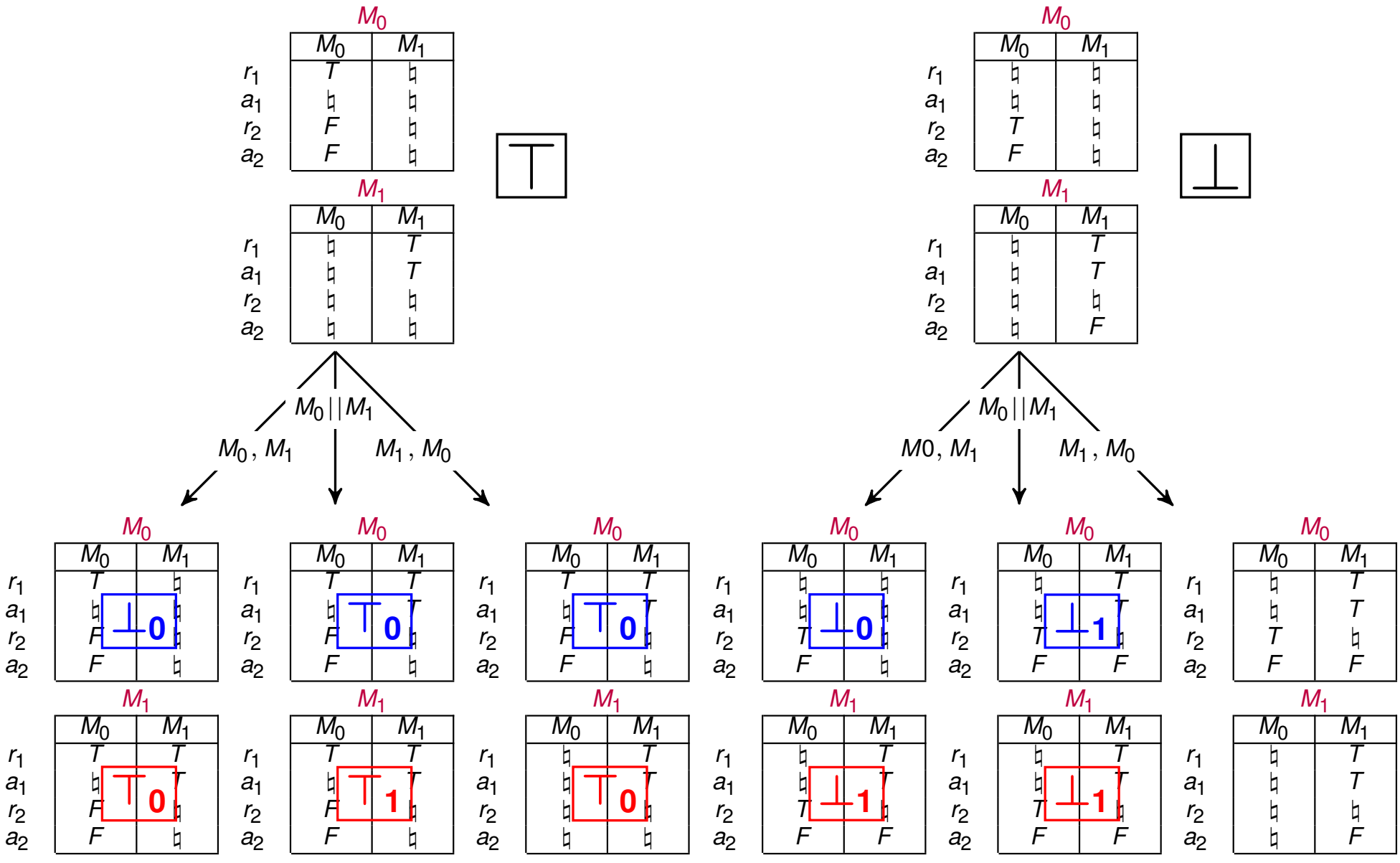

M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
0
0
0
0
1
0

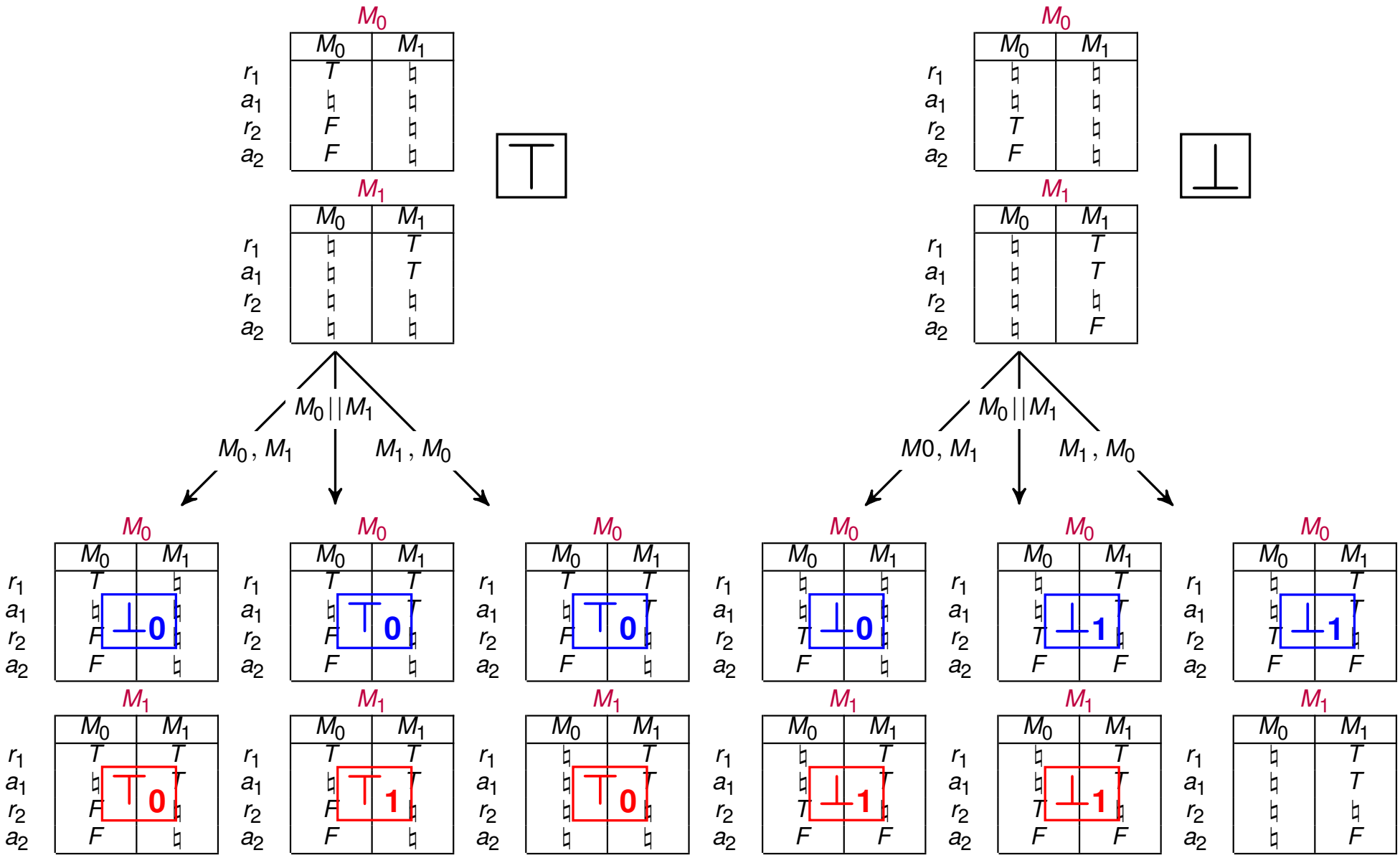

M_0
M_0 M_1
r_1 ↯ ↯
a_1 ↯ ↯
r_2 T ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ F

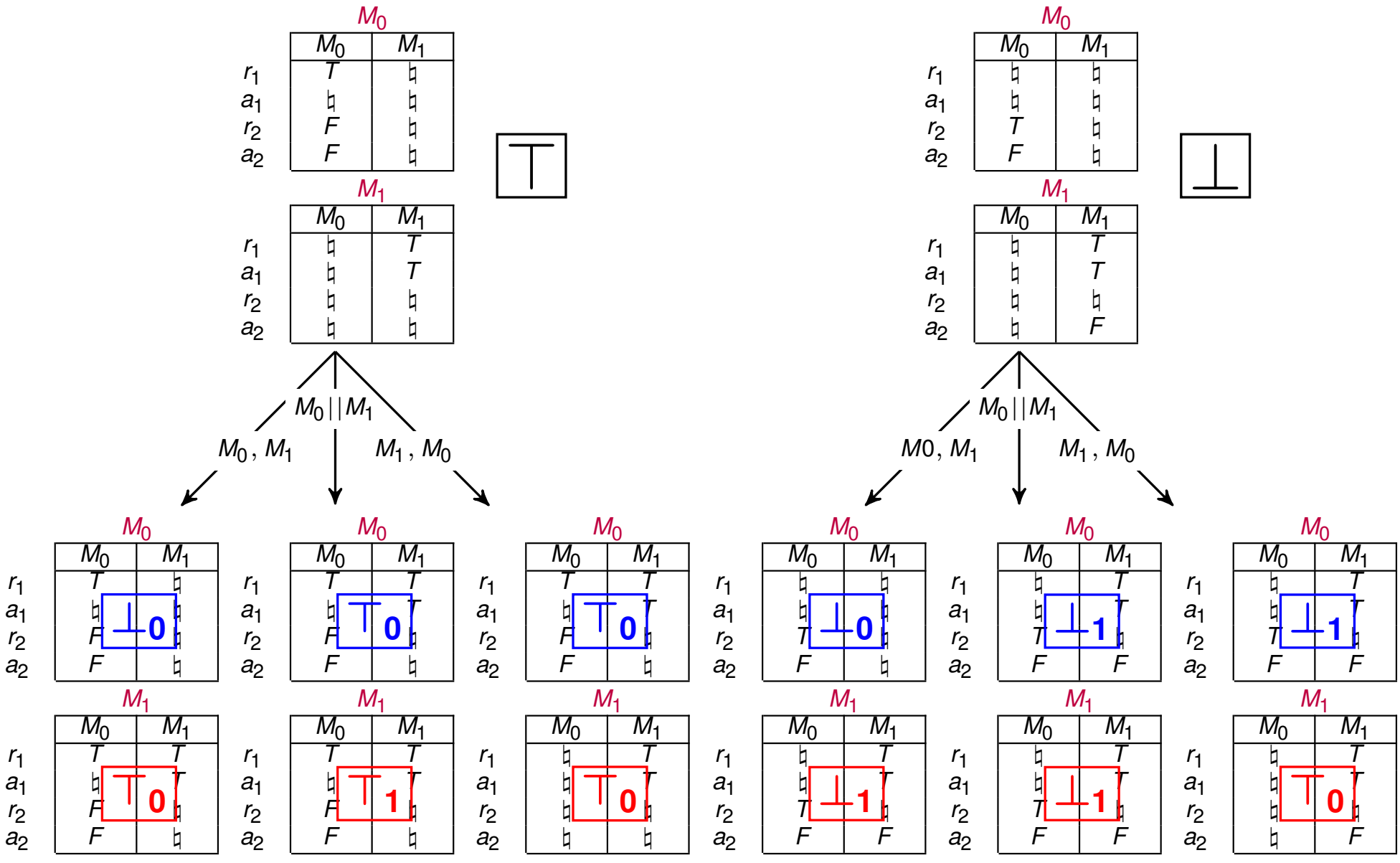

M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
M0, M1
0
1

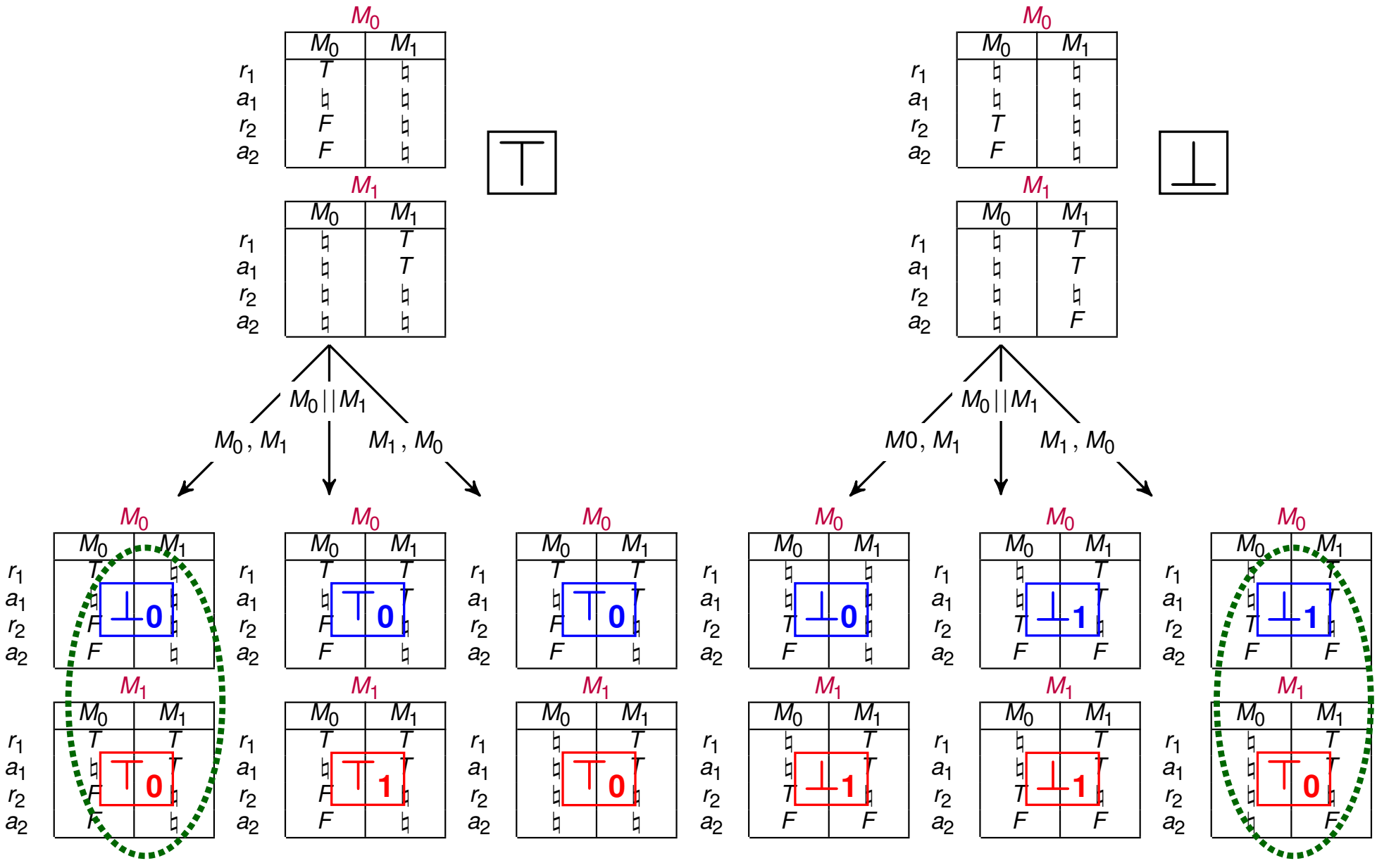

M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
M_0
M_0 M_1
r_1 T ↯
a_1 ↯ 0 ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ 0 T
r_2 F ↯
a_2 F ↯
M_0
M_0 M_1
r_1 T T
a_1 ↯ 0 T
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 T T
a_1 ↯ 1 T
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ 0 T
r_2 ↯ ↯
a_2 ↯ ↯
M_0
M_0 M_1
r_1 ↯ ↯
a_1 ↯ ↯
r_2 T ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ F
$M_0 || M_1$
$M0 , M_1$
M_1 , M_0
M_0
M_0 M_1
r_1 ↯ ↯
a_1 ↯ 0 ↯
r_2 T ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ 0 T
r_2 T ↯
a_2 F ↯
M_0
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 T ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 T ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ F

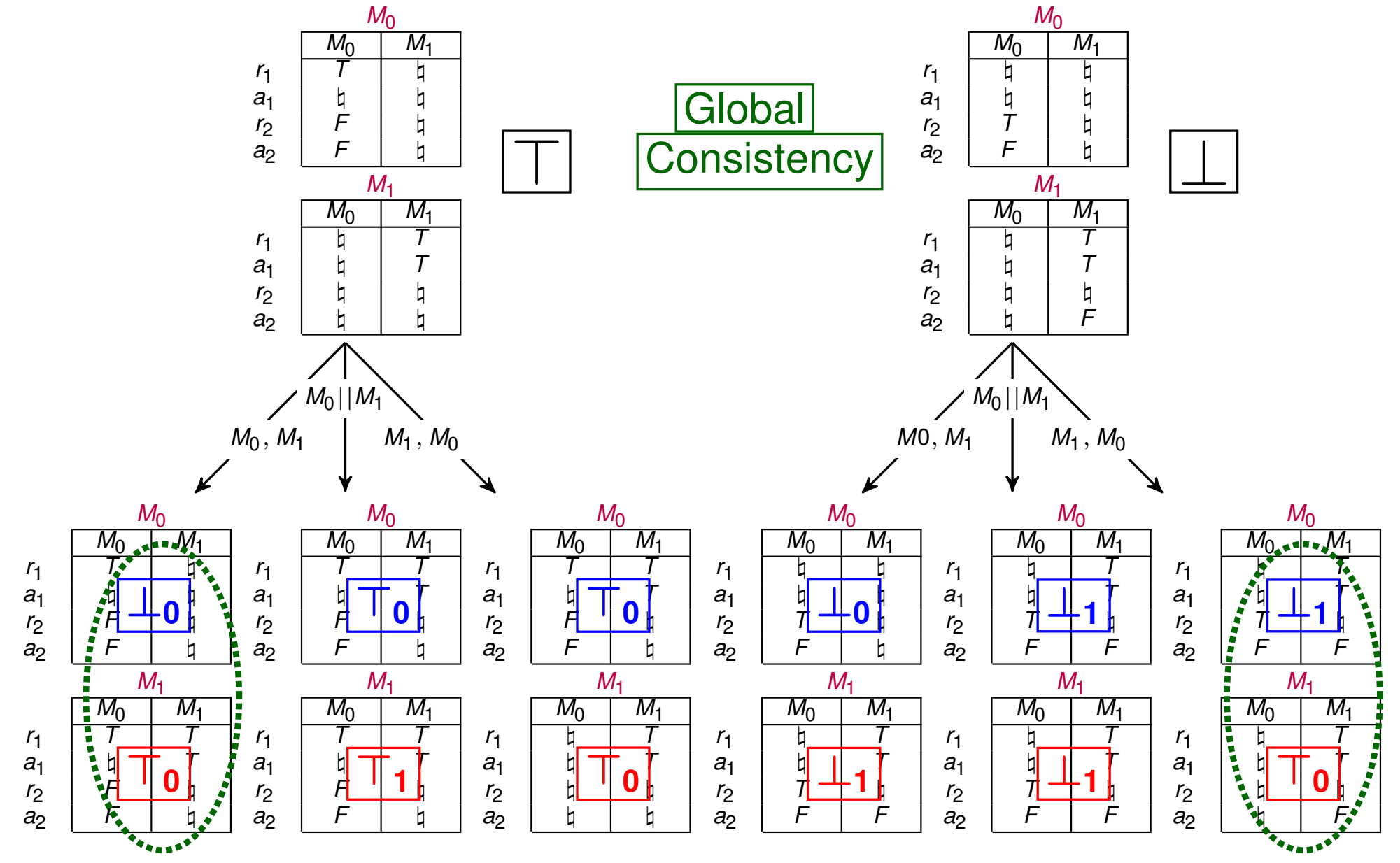

M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
$M0 , M_1$
0
1

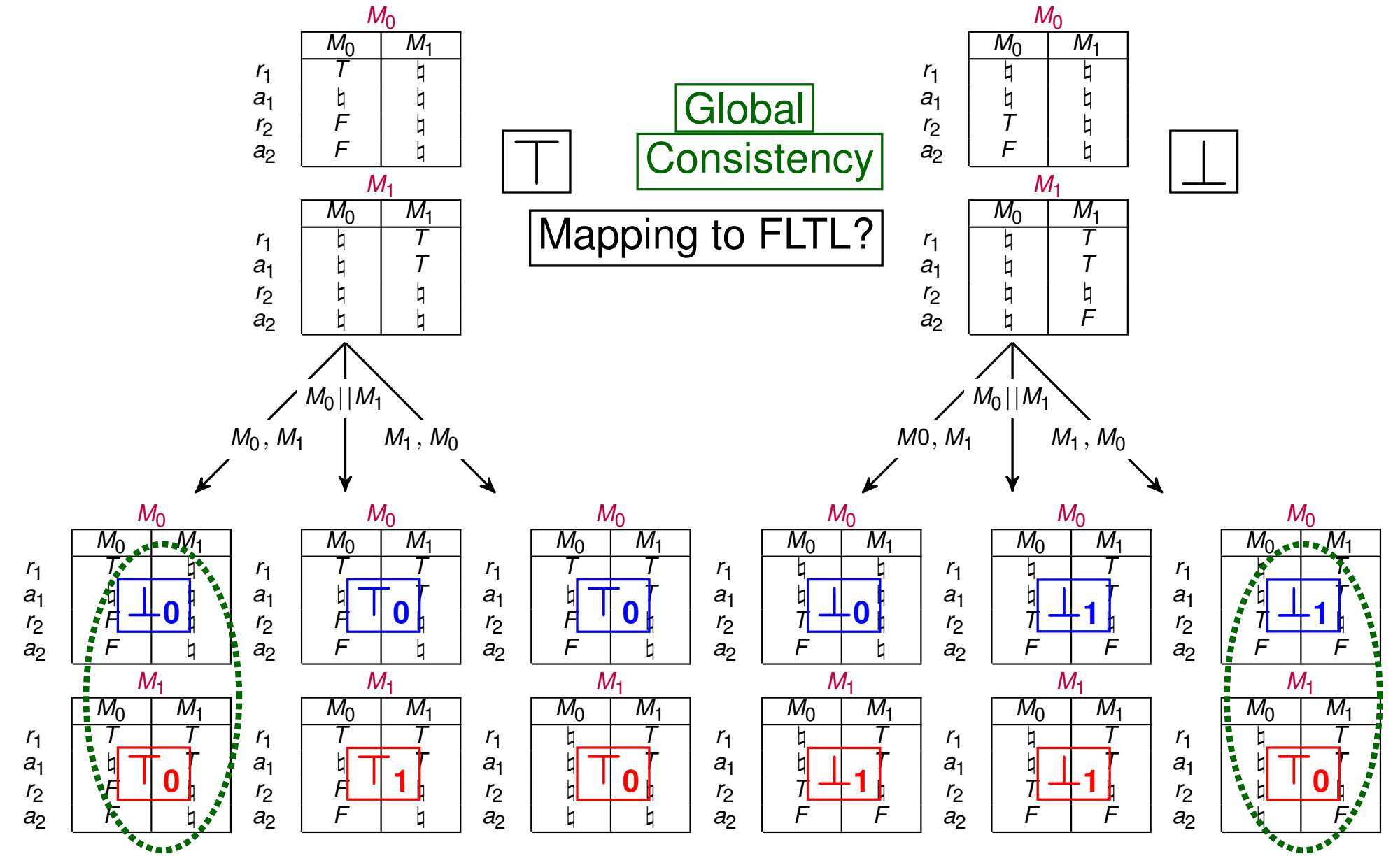

M_0
M_0 M_1
r_1 T ↯
a_1 ↯ ↯
r_2 F ↯
a_2 F ↯
M_1
M_0 M_1
r_1 ↯ T
a_1 ↯ T
r_2 ↯ ↯
a_2 ↯ ↯
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
0
0
0
0
1
1
0
1
1


M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
M0, M1
0
1


M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0, M_1
M_1, M_0
$M0, M_1$
0
1


M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
M0, M1
0
1


Global
Consistency
M_0
M_1
r_1
a_1
r_2
a_2
$M_0 || M_1$
M_0, M_1
M_1, M_0
M0, M1
0
1


Global
Consistency
Mapping to FLTL?
M_0
M_1
$M_0 || M_1$
M_0, M_1
M_1, M_0
$M0, M_1$
r_1
a_1
r_2
a_2
T
F
0
1

General Results

Theorem

An LTL formula φ can consistently be monitored by a wait-free distributed monitor in LTL_{2k+4}, if $2k + 2 \geq AN(\varphi)$.

Theorem

For each $k \geq 0$, there is an LTL formula φ that cannot be consistently monitored by a wait-free distributed monitor in LTL_{2k+4}, if $2k + 2 < AN(\varphi)$.

Presentation outline

Conclusion

Summary

This talk

- argued that existing RV logics are too abstract to monitor distributed systems in the presence of crash faults.
- introduced LTL_{2k+4} logic to overcome the problem
- proposed a monitor construction as well as an RV algorithm for LTL_{2k+4}.

Conclusion

Future Work

- Distributed monitoring when input propositions keep changing.
- Computing the bounds on alternation number in polynomial time.
- Lower/upper bounds for synchronous distributed monitors.
- Distributed monitoring of HyperLTL.
- Distributed monitoring in message passing system in the presence of Byzantine faults.
- Runtime enforcement of LTL properties in a distributed setting

McMaster University

Thank You!