The Power of Proofs: New Algorithms for Timed Automata Model Checking

Peter Fontana and Rance Cleaveland
Department of Computer Science
CMACS 2013: Nov 22, 2013

UNIVERSITY OF MARYLAND

Goal: Automatic Verification with Timing Constraints

Formally verify program correctness

Automate the verification

Handle **time and timing constraints**, both in model and specification

Timing Constraints Exist: Model Constraints

We allow the train to wait for different amounts of time

The gate takes time to lower

Timing Constraints Exist: Specification Constraints

The gate will be up within 2 minutes after a train leaves

Any train is in the region is in the region for at most 4 minutes

Our Framework

Programs modeled with **timed automata**

Properties specified with a **timed mu-calculus** (a modal logic)

Tool Implementation Exists

Peter Fontana and Rance Cleaveland. *On-The-Fly Timed Automata Model Checking*. Presented at CMACS PI Meeting on May 16, 2013

The Power of Proofs

This tool generates a **mathematical proof**

Verification using **proof rules**

We optimize performance by using **derived** proof rules

The Trick: Memoization

“Those who cannot remember the past are condemned to repeat it” (George Santayana)

The Trick: Memoization

Fibonacci Series: $a_0 = 1$, $a_1 = 1$, $a_n = a_{n-2} + a_{n-1}$

Compute a_4:

$a_4 = \mathbf{a_2} + a_3$

$\mathbf{a_2} = a_1 + a_0 = 1 + 1 = 2$

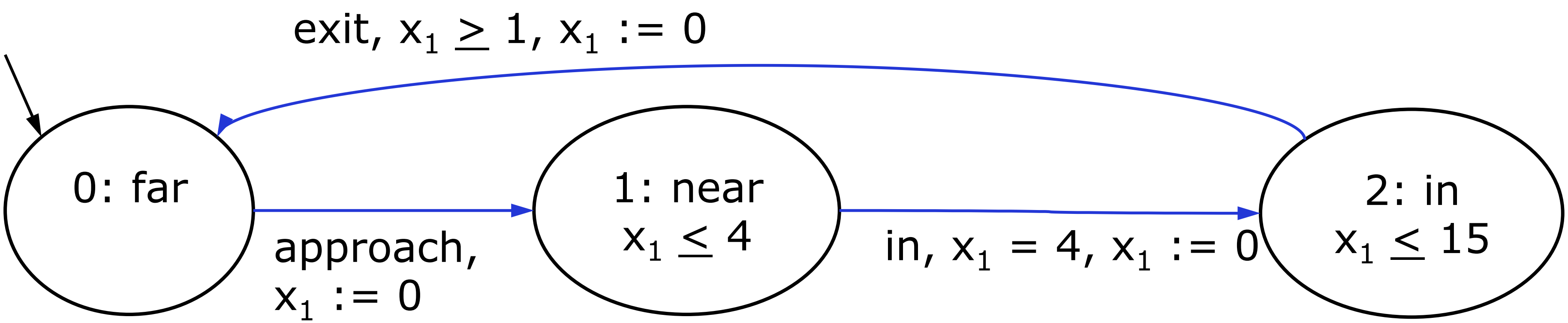
Memoization: Store "a2 = 2"

$a_4 = 2 + a_3$

$a_4 = 2 + (a_2 + a_1)$

The Details

Model: Timed Automata (State Machine + Clocks) [AD94]



Alur-Dill Model: timing constraints use clocks

A **state** is a **(location, clock values)** pair

Specification: Timed Modal Mu-Calculus $L^{rel}_{\nu,\mu}$

Boolean Logic

Variables X_i

Action Modalities $[a](\varphi)$, $\langle a\rangle(\varphi)$, $[-](\varphi)$, $\langle -\rangle(\varphi)$

Time Modalities $\forall(\varphi)$, $\exists(\varphi)$

Fixpoints $\overset{\nu}{=}$, $\overset{\mu}{=}$

Relativized Time Modalities $\forall_{\varphi_1}(\varphi_2)$, $\exists_{\varphi_1}(\varphi_2)$

Fixpoints

Definition (Formal): A **fixpoint** of a function f is a value x such that f(x) = x

The Power of Fixpoints: Writing Always Recursively

Always p: **p** is true now, and **Always p** is true in all next states.

$$X_1 \stackrel{\nu}{=} p \wedge \forall([-](X_1))$$

Note: This simplified formula assumes p only contains atomic propositions

The Power of Fixpoints: Formulas Represent States

Always p: **p** is true now, and **Always p** is true in all next states.

$$X_1 \stackrel{\nu}{=} p \wedge \forall([-](X_1))$$

X_1 is a **set of states** computed by this formula

Function f: $f(X_1) = p \wedge \forall([-](X_1))$

The Power of Fixpoints: Recursion as Local Search

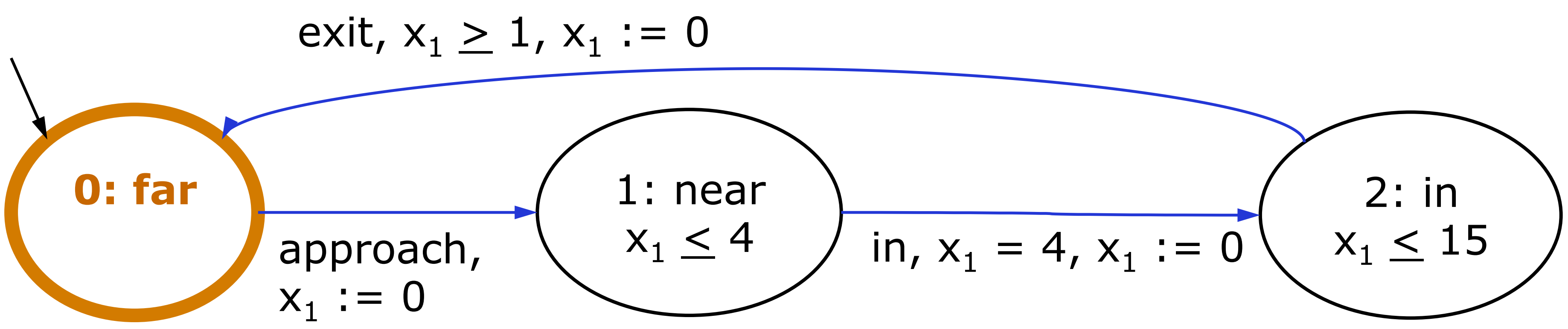
Always p: **p** is true now, and **Always p** is true in all next states.

$$X_1 \stackrel{\nu}{=} p \wedge \forall([-](X_1))$$

1. Have X_1 start at the initial state
2. Formula transitions X_1 to all next states
3. Stop when X_1 is a previously seen state

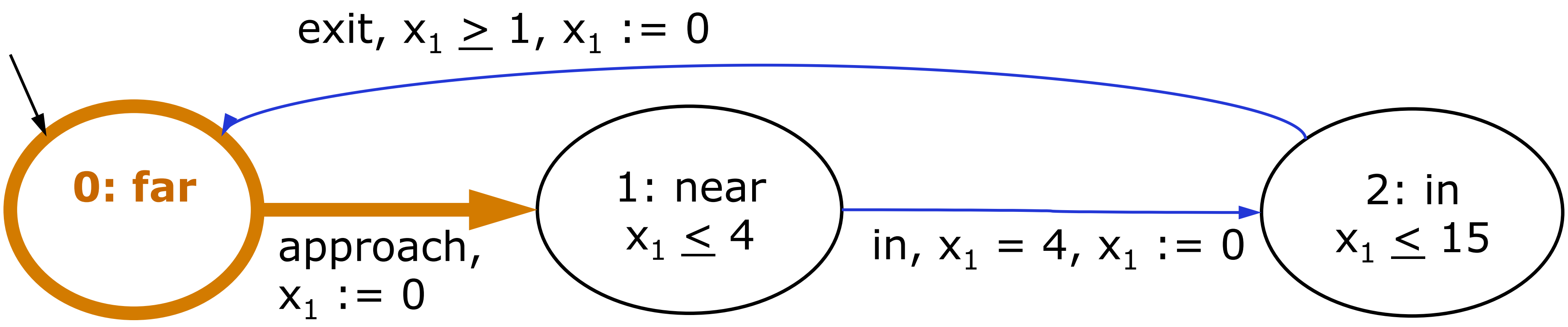
Greatest Fixpoint (ν): Visiting a previous state implies formula truth

The Power of Fixpoints: Never broken (AG)



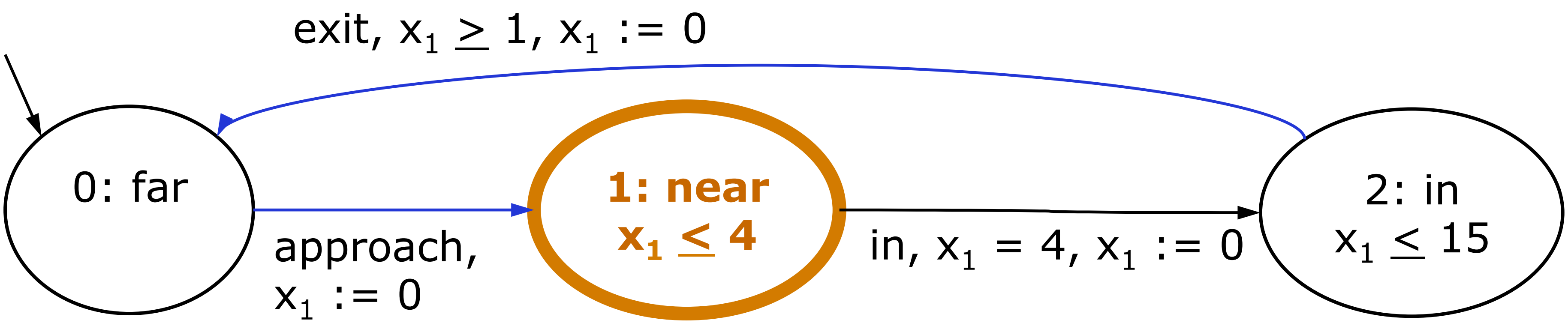
Verifier: Location **0: far** is not broken

The Power of Fixpoints: Never broken (AG)



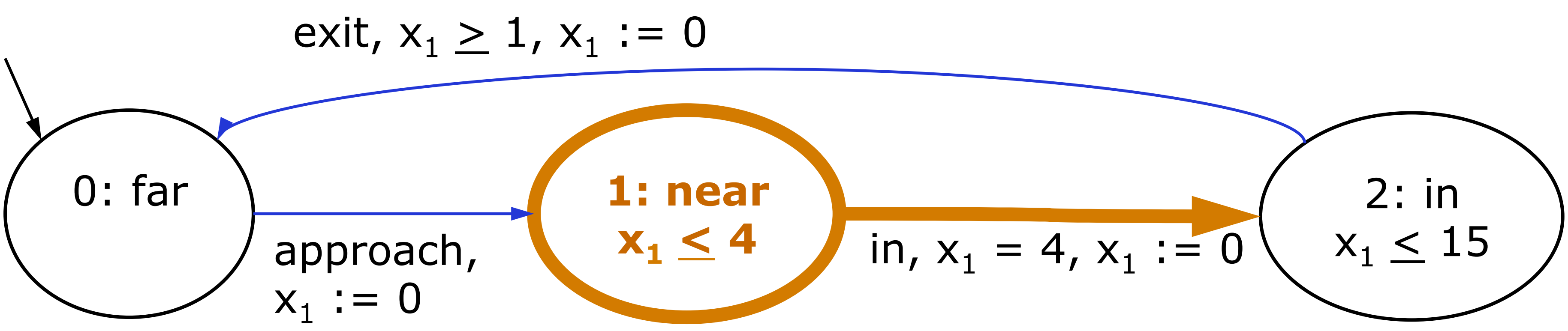
Verifier: Location **0: far** is not broken

The Power of Fixpoints: Never broken (AG)



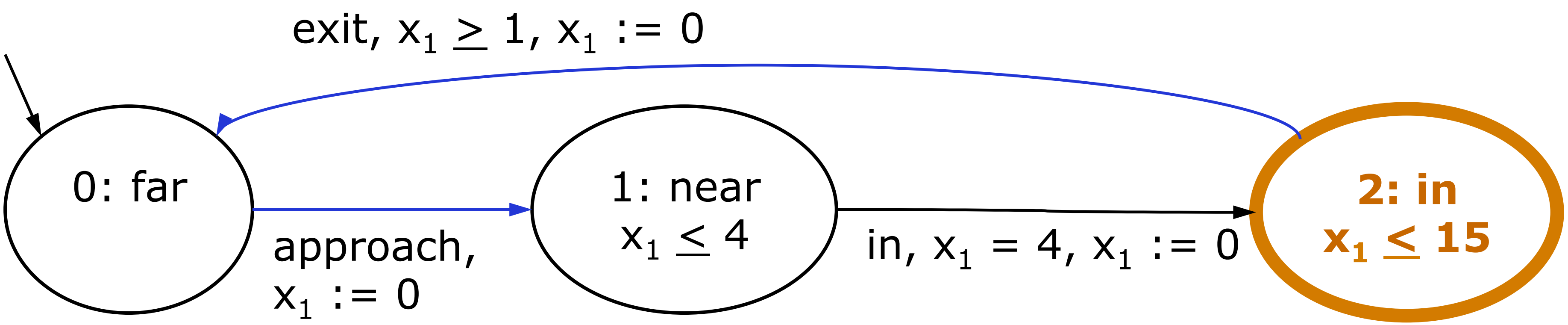
Verifier: Location **1: near** is not broken

The Power of Fixpoints: Never broken (AG)



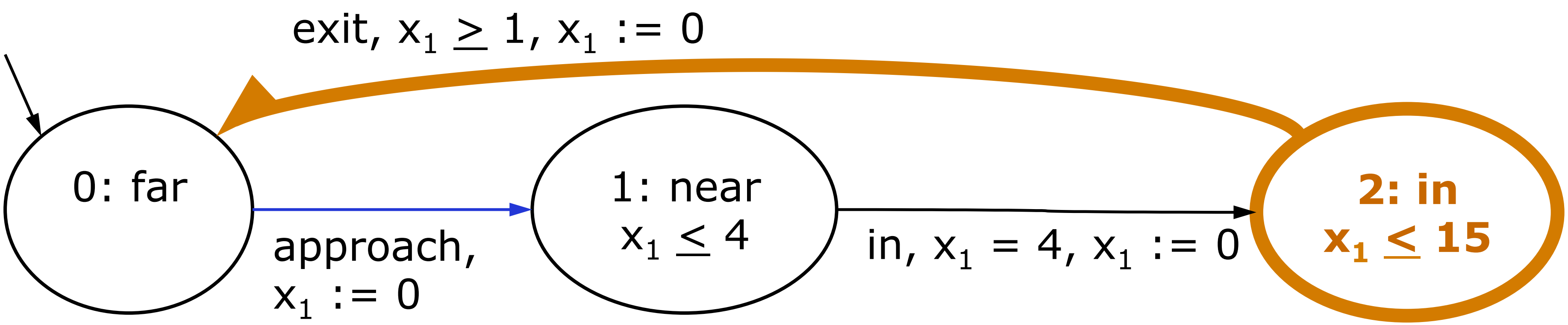
Verifier: Location **1: near** is not broken

The Power of Fixpoints: Never broken (AG)



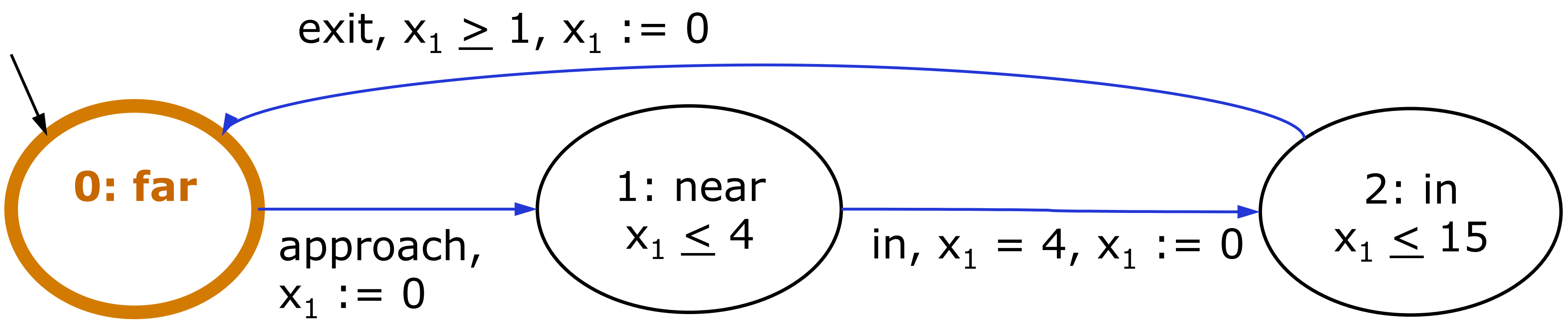
Verifier: Location **2: in** is not broken

The Power of Fixpoints: Never broken (AG)



Verifier: Location **2: in** is not broken

The Power of Fixpoints: Never broken (AG)



Verifier: We have visited **0: far** again (circularity); apply **greatest** fixpoint

Proof Rules: One Step at A Time (X_1: Always not broken)

$$\frac{\text{Premise } 1 \quad \ldots \quad \text{Premise } n}{\text{Conclusion}} \; (\textit{Rule Name})$$

$$\cfrac{\cfrac{\cfrac{\cfrac{(0 : \mathit{far}, \{x_1 = 0\}) \vdash X_1 \quad \textbf{True} \text{ (Greatest fixpont)}}{\ldots}}{(1 : \mathit{near}, \{x_1 = 0\}) \vdash X_1}}{(0 : \mathit{far}, \{x_1 = 0\}) \vdash \neg \mathit{broken} \wedge \text{all next states } X_1}}{(0 : \mathit{far}, \{x_1 = 0\}) \vdash X_1}$$

Relativization Operators

Definition: $L^{rel}_{v,\mu}$ relativization operators are:

$\exists_{\varphi_1}(\varphi_2)$: for all times $\delta' < \delta$, φ_1 is true

$\forall_{\varphi_1}(\varphi_2)$: φ_1 releases φ_2 from being true

Definition by **duality**: $\exists_{\varphi_1}(\varphi_2) \overset{def}{\equiv} \neg\forall_{\neg\varphi_1}(\neg\varphi_2)$

Obtaining $L_{v,\mu}$ operators: $\exists_{\mathtt{tt}}(\varphi), \forall_{\mathtt{ff}}(\varphi)$

Relativization Operators give Expressive Power

Theorem: We can express all of TCTL in $L^{rel}_{v,\mu}$

Relativization Operators?!? We Need Them!

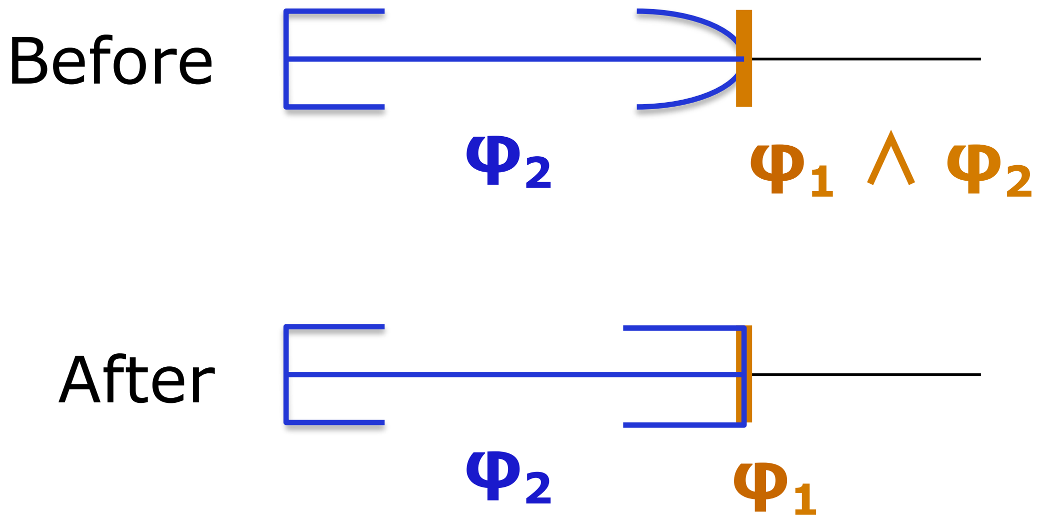
Theorem: We cannot express TCTL formula $A\varphi_1 R \varphi_2$ in $L_{\nu,\mu}$

Proof Rule Optimization 1: Relativized All

Lemma: $\forall_{\varphi_1}(\varphi_2) \equiv \forall(\varphi_2) \lor \exists_{\varphi_2}(\varphi_1 \land \varphi_2)$

Use proof of derivation to generate a **derived rule**

Relativized All Optimization: Rewrite a Subrule



$$\dfrac{\dfrac{\forall(\varphi_2) \vee \exists_{\leq\varphi_2}(\varphi_1)}{\forall(\varphi_2) \vee \exists_{\varphi_2}(\varphi_1 \wedge \varphi_2)}}{\forall_{\varphi_1}(\varphi_2)}$$

Relativized All Optimization: Memoize φ_2

$$\forall(\boxed{\varphi_2}) \vee \exists_{\leq \boxed{\varphi_2}}(\varphi_1)$$

1. Find **all states** that satisfy φ_1
2. Find **all states** that satisfy φ_2
3. Reason with **memoized** stored states to handle logic operators $\forall$, $\exists$

Correctness of Proof Rules

Theorem: The proof rules (original and derived) are **sound** and **complete**.

Conclusion

Implementation can check more specifications: the entire alternation-free fragment of L^{rel}

Using derived proof rules optimizes performance

Future Work

Further Proof Utilization: Extra verification information

Performance optimization

QED □