Refactoring, Refinement, and Reasoning

A Logical Characterization for Hybrid Systems

Stefan Mitsch[1,2] Jan-David Quesel[1] André Platzer[1]

[1]Computer Science Department, Carnegie Mellon University

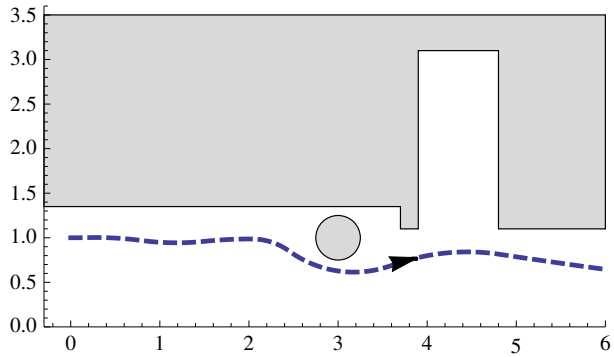
[2]Cooperative Information Systems, Johannes Kepler University

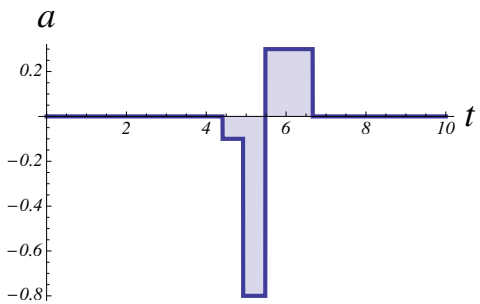
May 14, 2014

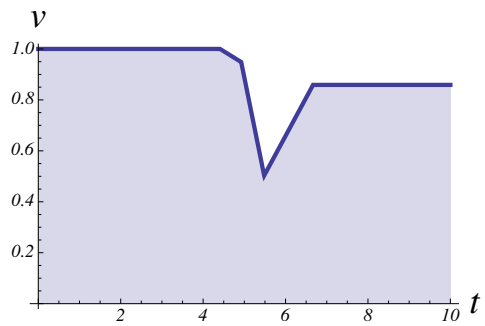
Hybrid Systems

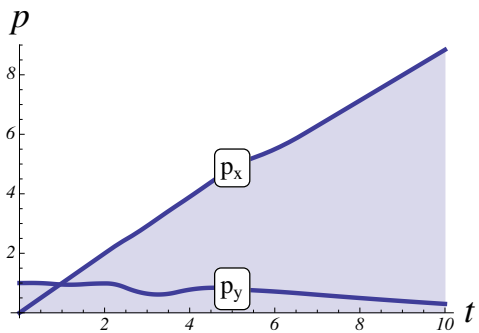
Hybrid Systems are Challenging

- Computation + Physical behavior
- Sensor uncertainty
- Disturbance
- Computation delay
- Many components





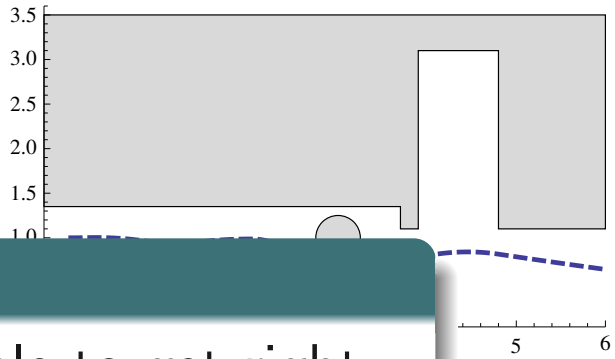




Hybrid Systems

Hybrid Systems are Challenging

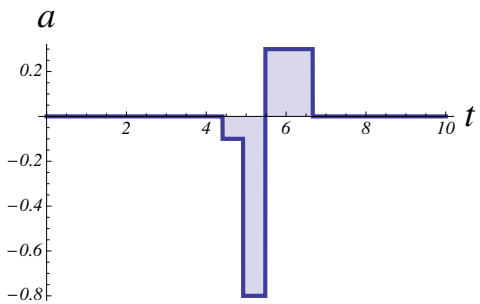
- Computation + Physical behavior
- Sen
- Dist
- Con
- Man

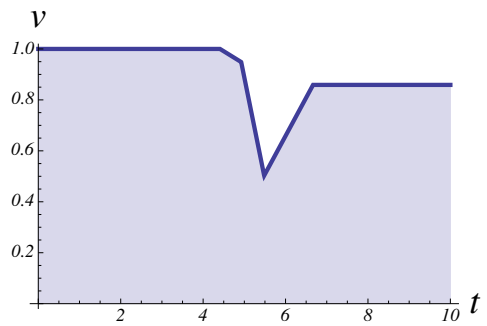


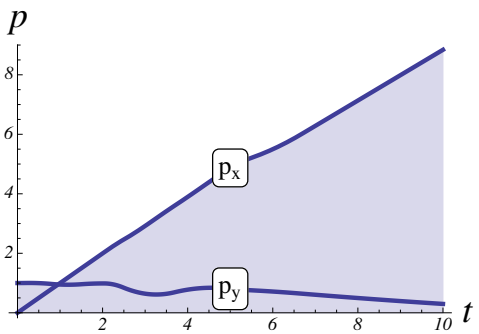
Challenge

Hybrid systems are almost impossible to get right without proper analysis

$\rightsquigarrow$ **Formal verification**







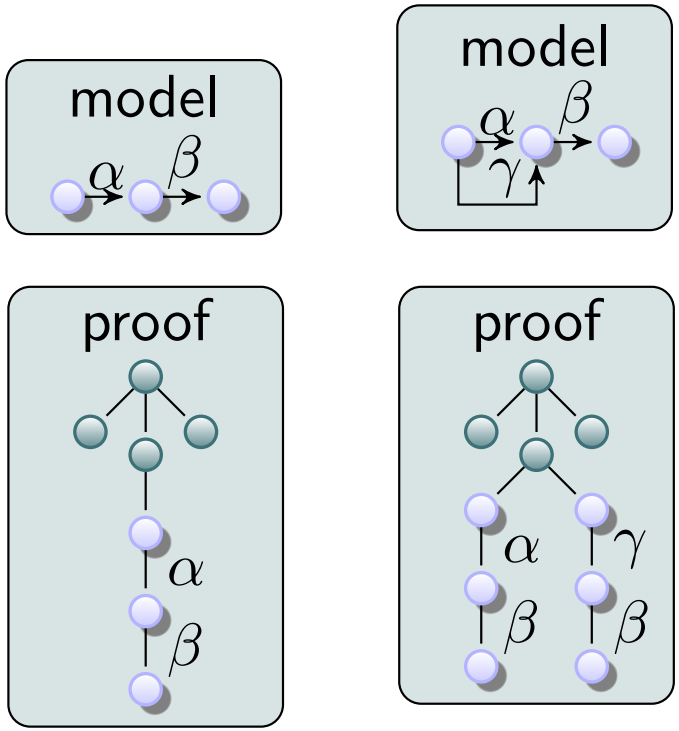
Formal Verification

Hybrid System Theorem Proving

- Symbolic execution of model
- Model structure reflected in proof
- Correctness properties

Safety Always stay safe

Liveness Ultimately complete a task



Our Tools

KeYmaera Hybrid systems theorem prover

Sφnx Hybrid systems modeling

Iterative Development

Hybrid Systems Theorem Proving is Challenging

- Differential equations
- Complicated arithmetic

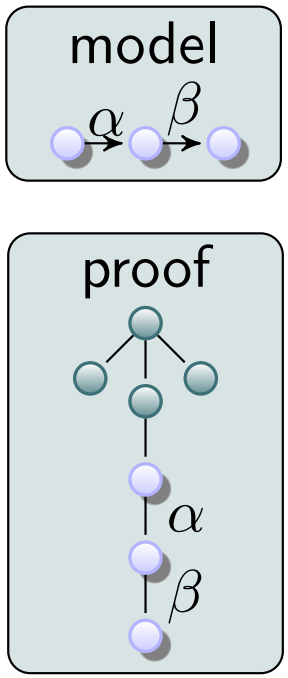
Iterative Development

Hybrid Systems Theorem Proving is Challenging

- Differential equations
- Complicated arithmetic

Manage complexity

- Start simple — verify



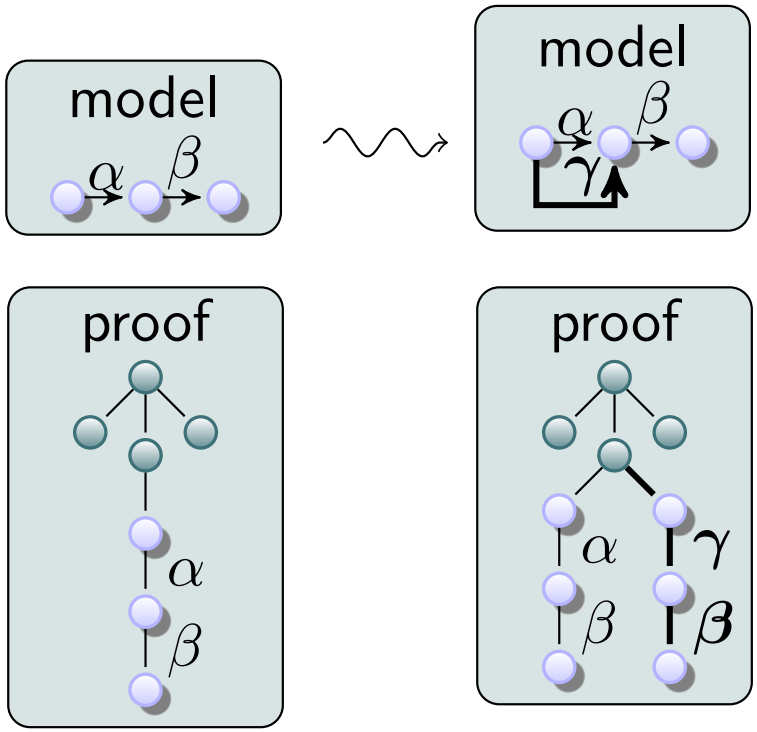
Iterative Development

Hybrid Systems Theorem Proving is Challenging

- Differential equations
- Complicated arithmetic

Manage complexity

- Start simple — verify
- Improve — verify — repeat



Iterative Development

Hybrid Systems Theorem Proving is Challenging

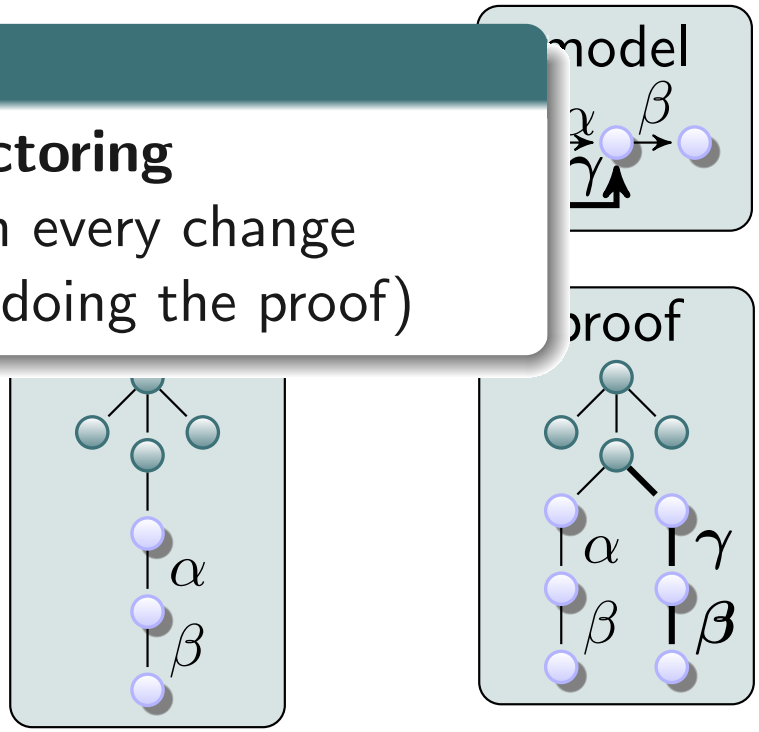
- Differential equations
- C

Challenge

Proof-aware refactoring
instead of reverification on every change
(retain soundness without redoing the proof)

Mana

- St
- Improve — verify — repeat



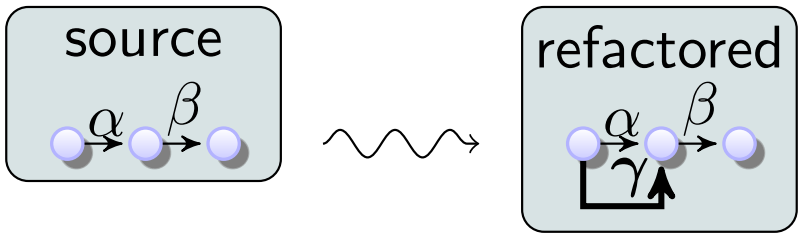
Proof-aware Refactoring

Refactoring Operation

- Transforms a source model into a refactored model
- Syntactic rewriting rule

For example

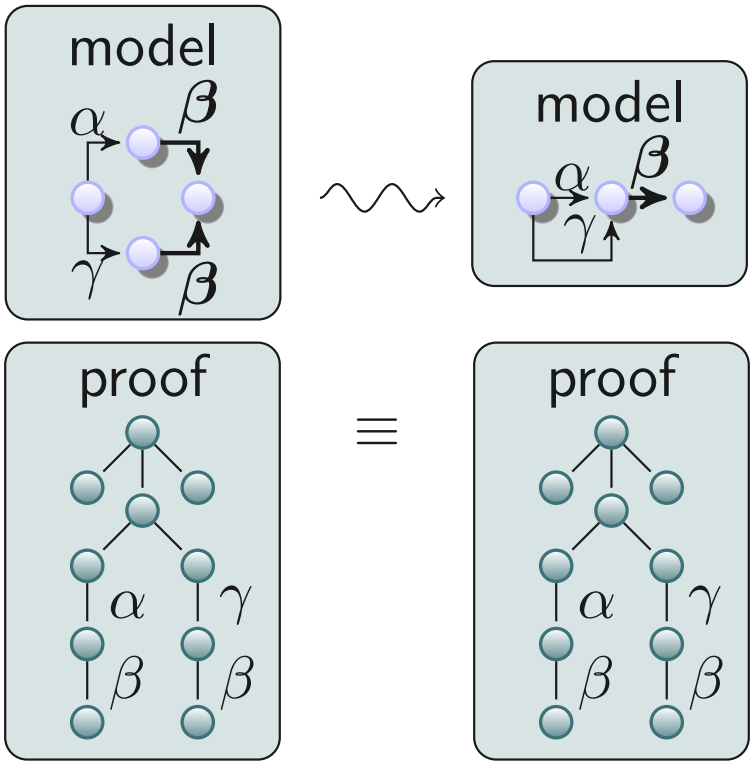
$$\frac{\text{conditions}}{\alpha;\beta \rightsquigarrow (\alpha \cup \gamma);\beta}$$



How to Retain Soundness

Structural Refactoring

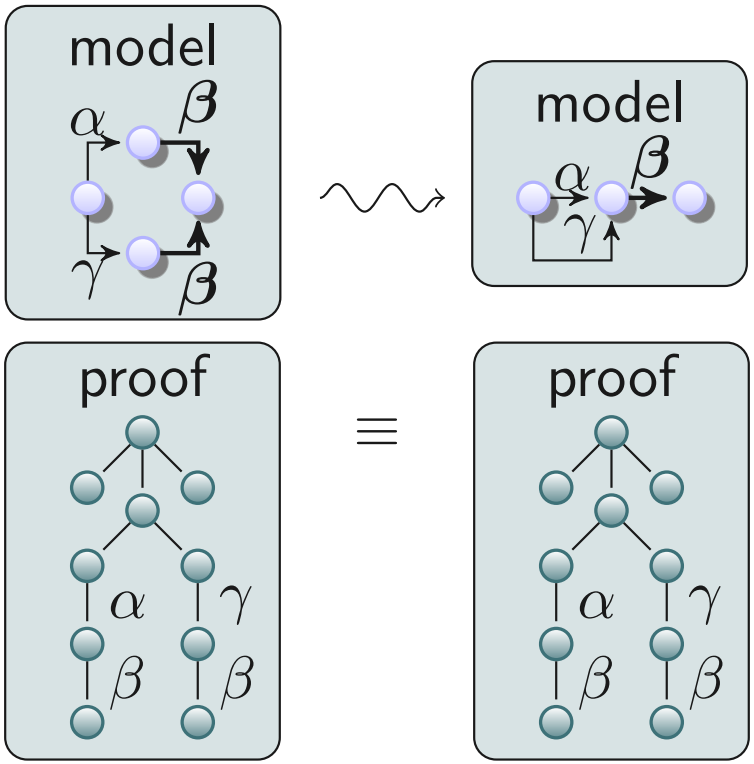
Always retains soundness



How to Retain Soundness

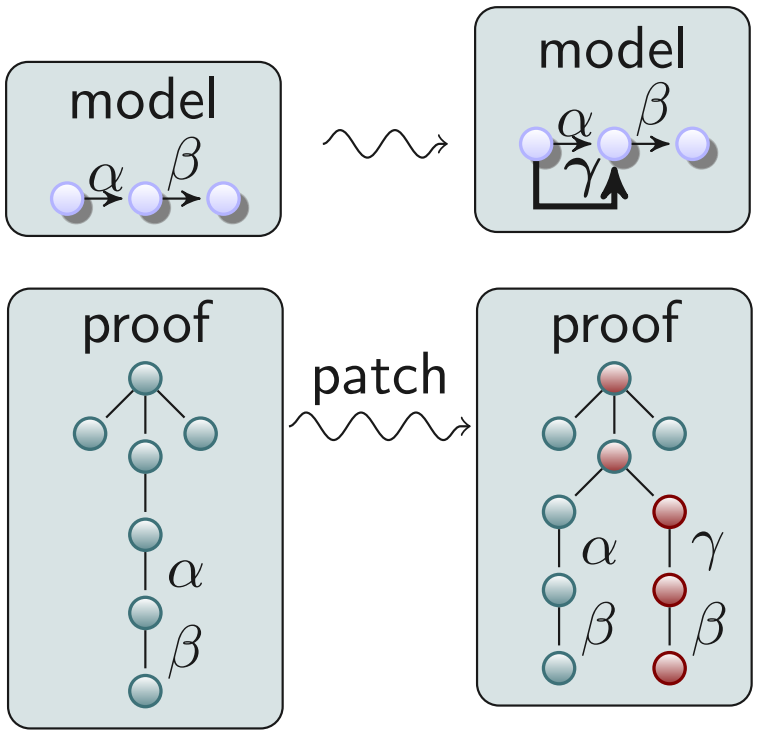
Structural Refactoring

Always retains soundness



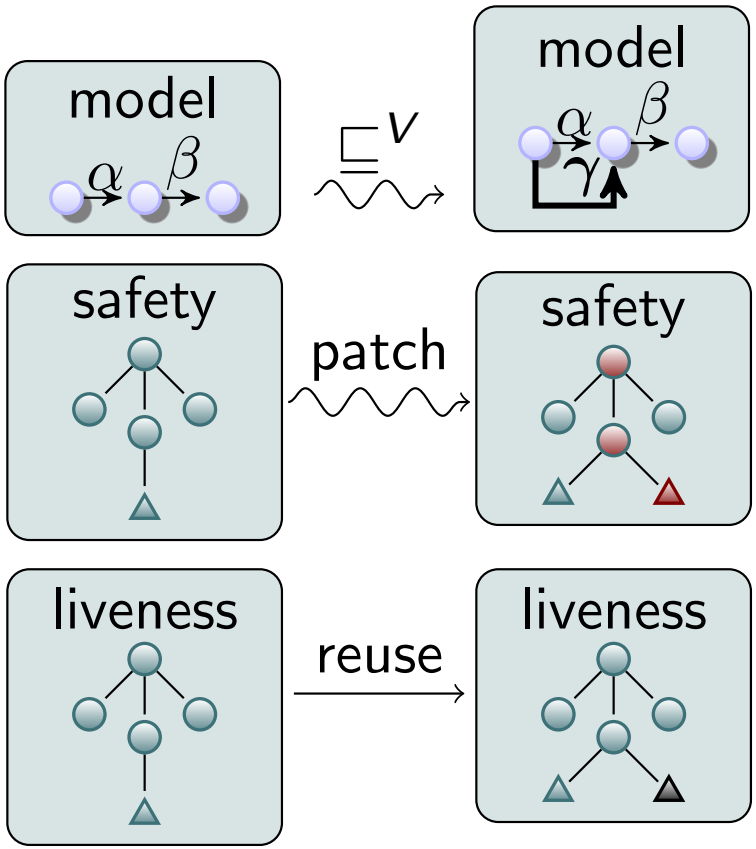
Behavioral Refactoring

Proof patch retains soundness



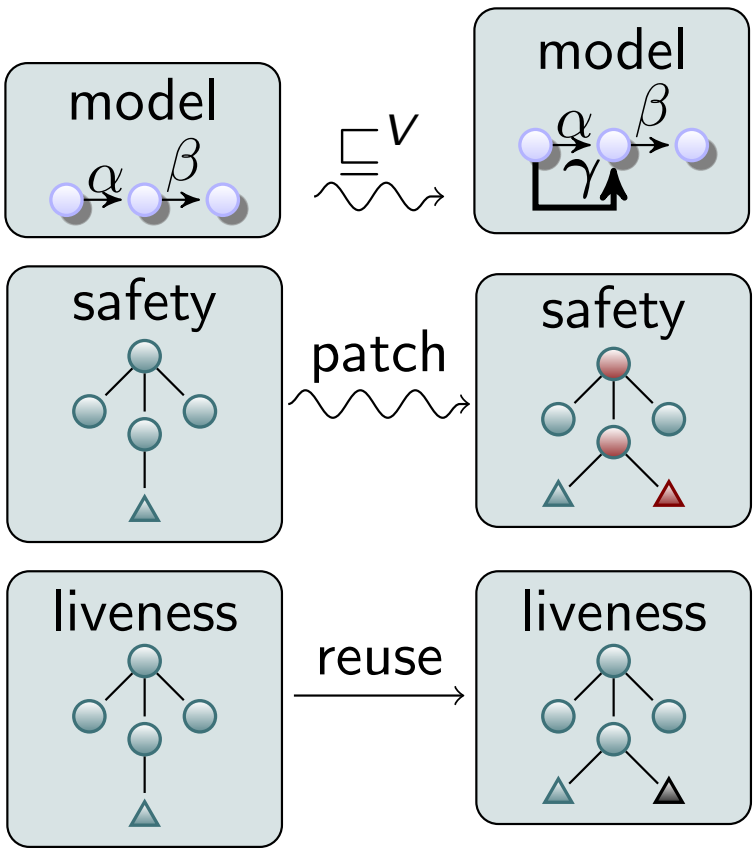
Patching Necessity by Correctness Property

Add Behavior

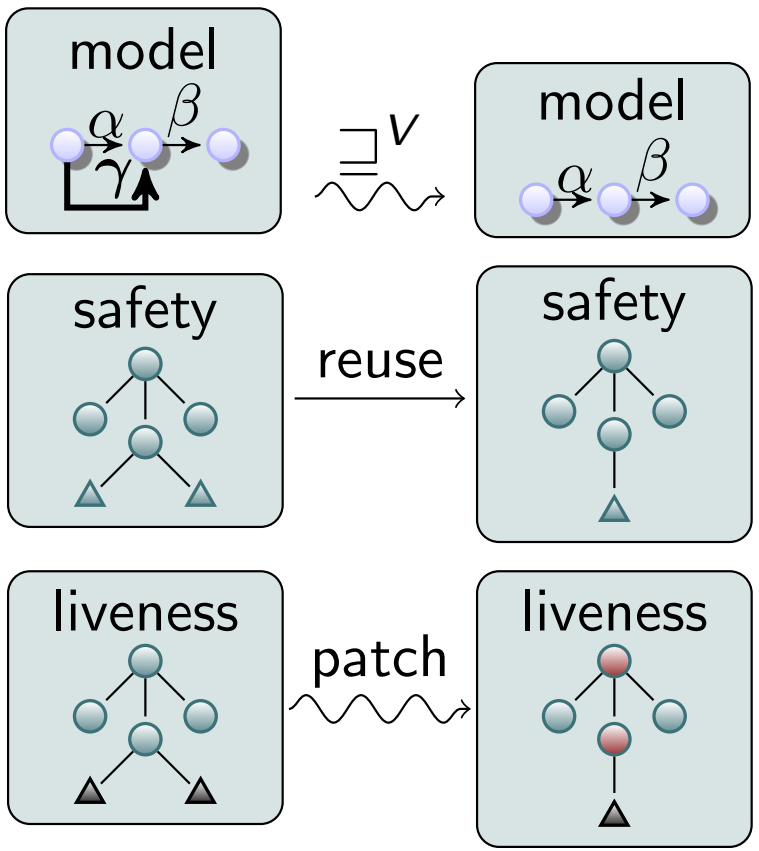


Patching Necessity by Correctness Property

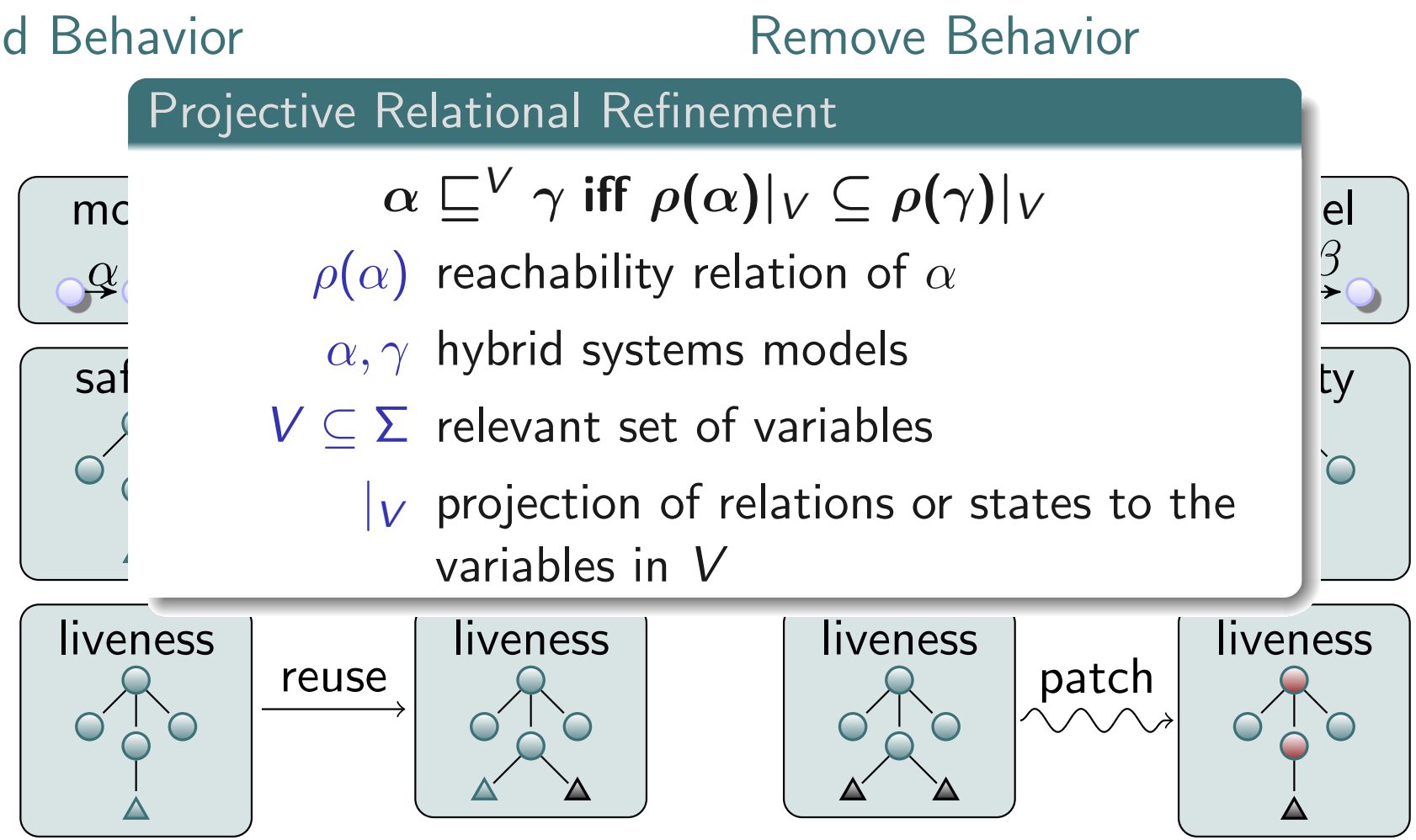
Add Behavior



Remove Behavior



Patching Necessity by Correctness Property



Sound Refactoring Catalog

Structural Refactorings

- Extract Common Program
- Extract Continuous Dynamics
- Drop Implied Evolution Domain Constraint

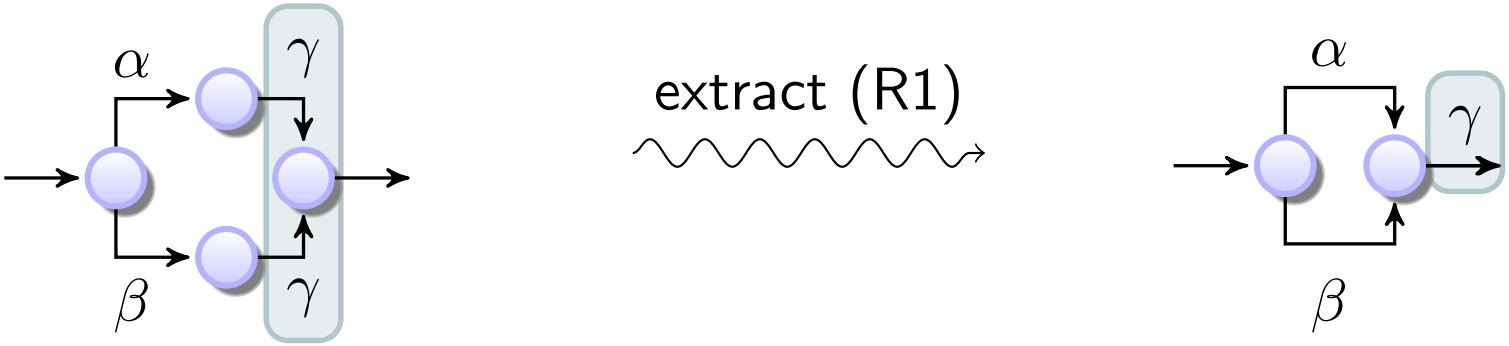
Behavioral Refactorings

- Introduce Control Path
- Introduce Complementary Continuous Dynamics
- Event- to Time-Triggered Architecture

Extract Common Program

Motivation Reduce model duplication

Mechanics (R1) $(\alpha;\gamma)\cup(\beta;\gamma) \rightsquigarrow (\alpha\cup\beta);\gamma$



Variation Inline program (R2) $(\alpha\cup\beta);\gamma \rightsquigarrow (\alpha;\gamma)\cup(\beta;\gamma)$

Proof patch

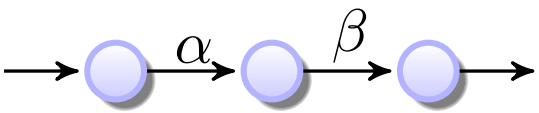
Safety None

Liveness None

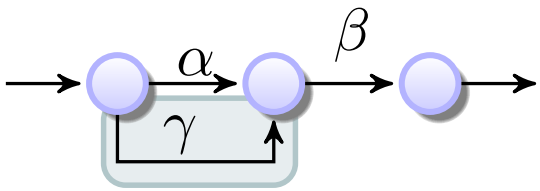
Introduce Control Path

Motivation Add control decisions

Mechanics (R3) $\alpha;\beta \rightsquigarrow (\alpha \cup \gamma);\beta$



introduce (R3) $\rightsquigarrow$



Variation Remove Control Path (R4) $(\alpha \cup \gamma);\beta \rightsquigarrow \alpha;\beta$

Proof patch

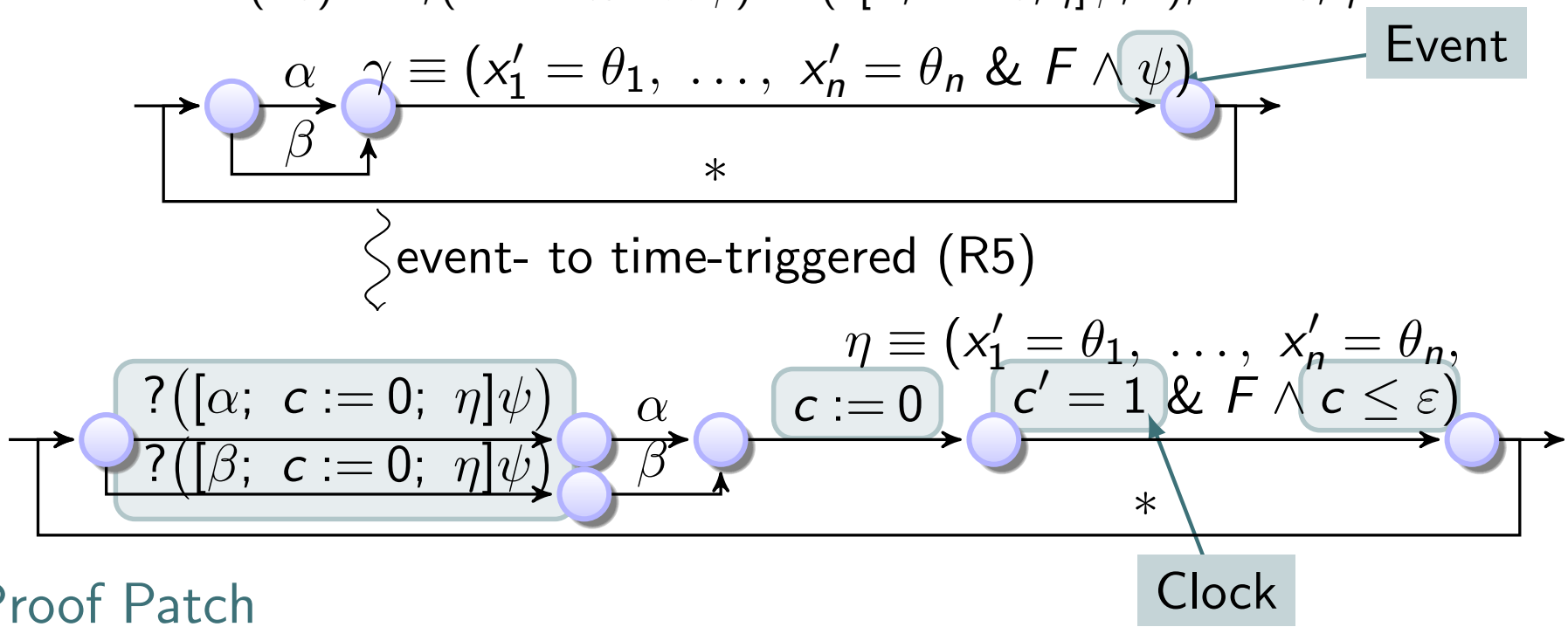
Safety Prove safety of the added branch

Liveness None

Event- to Time-Triggered Architecture

Motivation Derive a time-triggered controller

Mechanics (R5) $\alpha; (x' = \theta \,\&\, F \wedge \psi) \rightsquigarrow (?[\alpha; c := 0; \eta]\psi; \alpha); c := 0; \eta$



Proof Patch

Safety Composes several refactorings + prove safety of derived tests

Summary

Benefits of Proof-aware Refactorings

- easier to evolve correct systems
- easier to get simple systems correct
- still want to handle complex systems, but not pay the price of reverification
- **co-evolve model and proof**

Future Work

Refactoring Catalog

- Pull and merge tests
- Weaken/strengthen test
- Switch sequence
- Introduce computation delay
- Introduce uncertainty
- Introduce disturbance
- Change norm ($2/\infty$ norm)
- ...

Theory

- Liveness proof patches
- Distance measurement
- Refinement based on games

Implementation

- Sφnx and KeYmaera
- Background proving

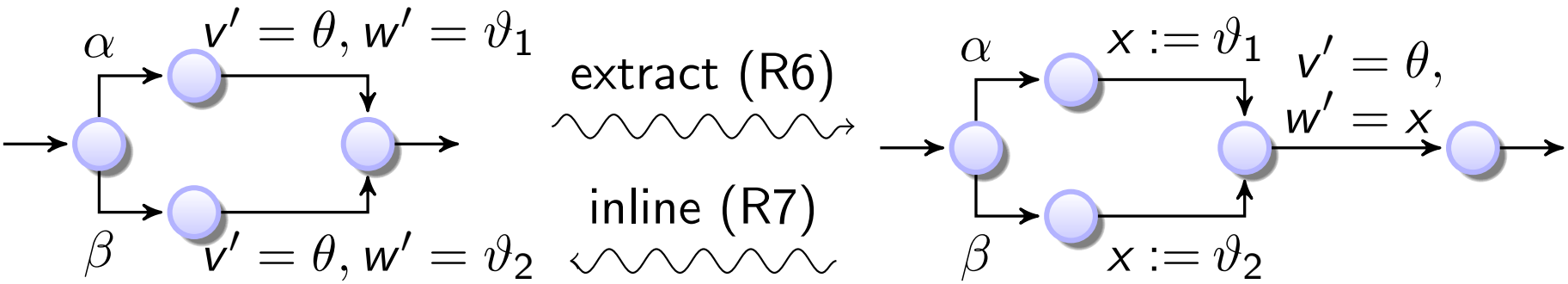
Thank you!

Stefan Mitsch
smitsch@cs.cmu.edu
http://www.cs.cmu.edu/∼smitsch

Extract Continuous Dynamics

$$\text{(R6)}\quad \frac{\forall v \in V(\theta) \cup \bigcup_{i\in I} V(\vartheta_i).\ v \notin BV(\mathcal{D}(\theta)) \cup \bigcup_{i\in I} BV(\mathcal{D}(\vartheta_i))}{\bigcup_{i\in I}(\alpha_i; (v' = \theta, w' = \vartheta_i)) \rightsquigarrow \left(\bigcup_{i\in I}(\alpha_i; x := \vartheta_i)\right); (v' = \theta, w' = x)}$$

$$\text{(R7)}\quad \left(\bigcup_{i\in I}(\alpha_i; x := \vartheta_i)\right); (v' = \theta, w' = x) \rightsquigarrow \bigcup_{i\in I}(\alpha_i; (v' = \theta, w' = \vartheta_i))$$



Proof obligations

None, because the original program and the refactored program are **observationally equivalent**.

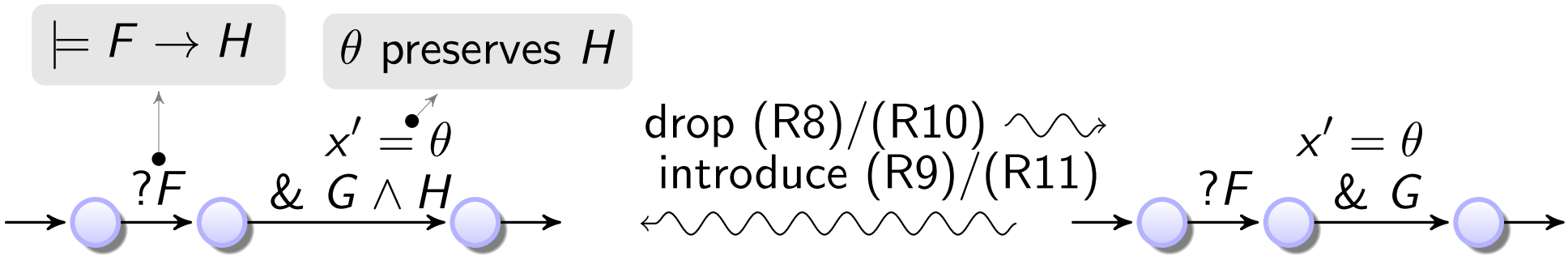
Drop Implied Evolution Domain Constraint

(R8) $\dfrac{F \to H \qquad F \to [x' = \theta \,\&\, G]H}{?F; x' = \theta \,\&\, G \wedge H \rightsquigarrow ?F; x' = \theta \,\&\, G}$

(R9) $?F; x' = \theta \,\&\, G \rightsquigarrow ?F; x' = \theta \,\&\, G \wedge H$

(R10) $?F; x' = \theta \,\&\, G \wedge H \rightsquigarrow ?F; x' = \theta \,\&\, G$

(R11) $\dfrac{F \to H \qquad F \to [x' = \theta \,\&\, G]H}{?F; x' = \theta \,\&\, G \rightsquigarrow ?F; x' = \theta \,\&\, G \wedge H}$



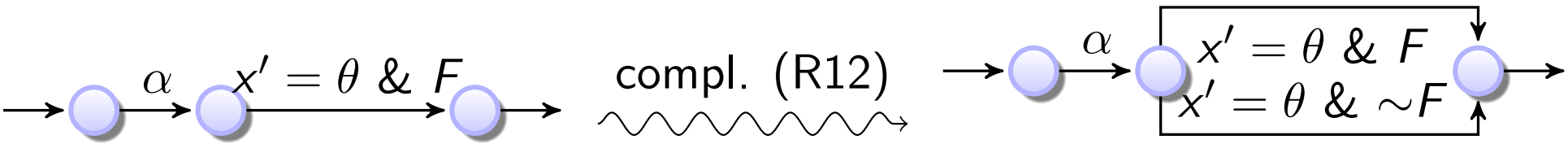
Proof obligations

Liveness None, because projective partial refinement, i. e., $\left(?F; x' = \theta \,\&\, G \wedge H\right) \sqsubseteq_F^V \left(?F; x' = \theta \,\&\, G\right)$ holds.

Safety Show that H is a differential invariant

Introduce Complementary Continuous Dynamics

(R12) $\alpha; x' = \theta \,\&\, F \;\rightsquigarrow\; \alpha; (x' = \theta \,\&\, F \cup x' = \theta \,\&\, \sim F)$

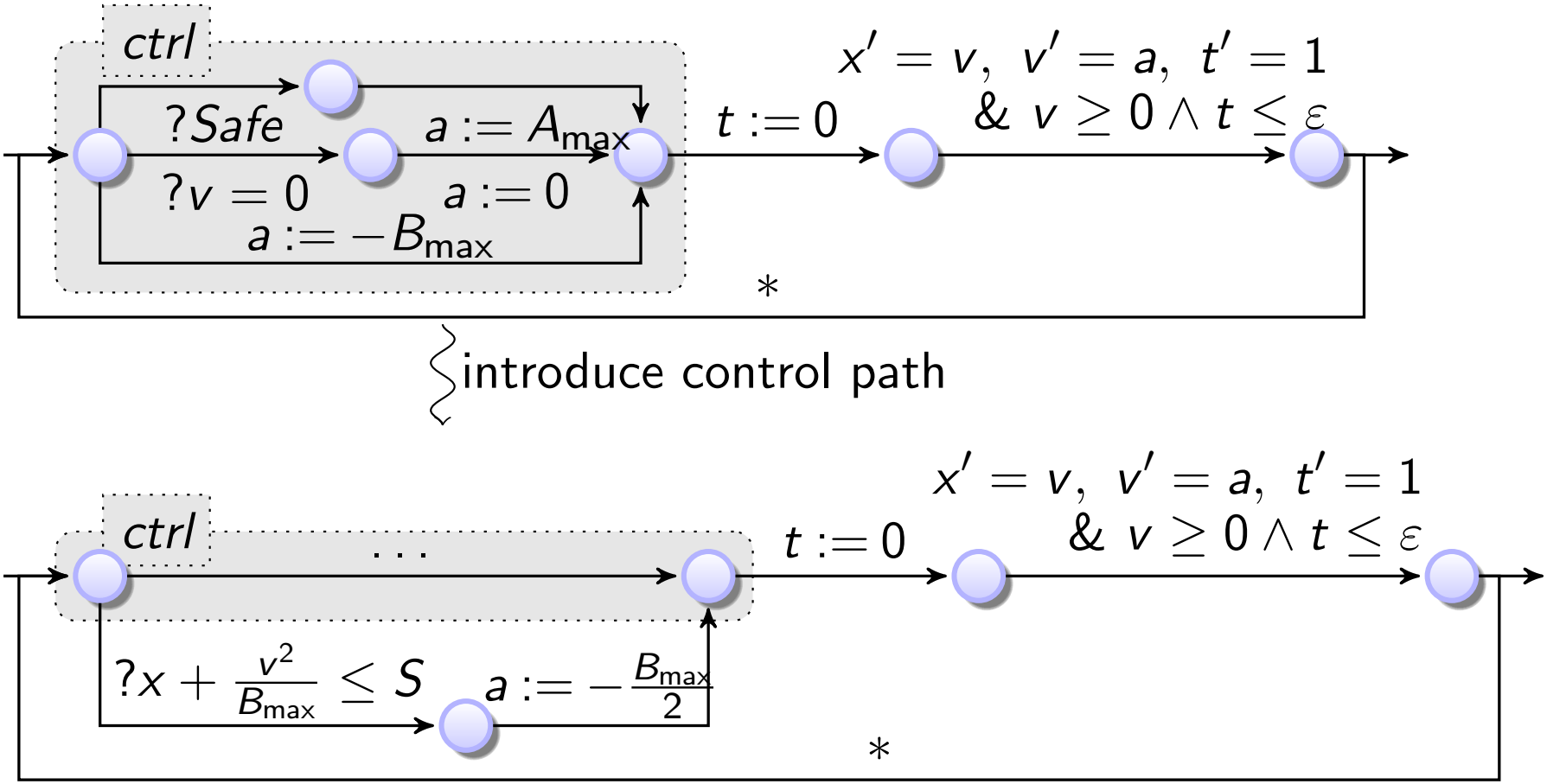


Proof obligations

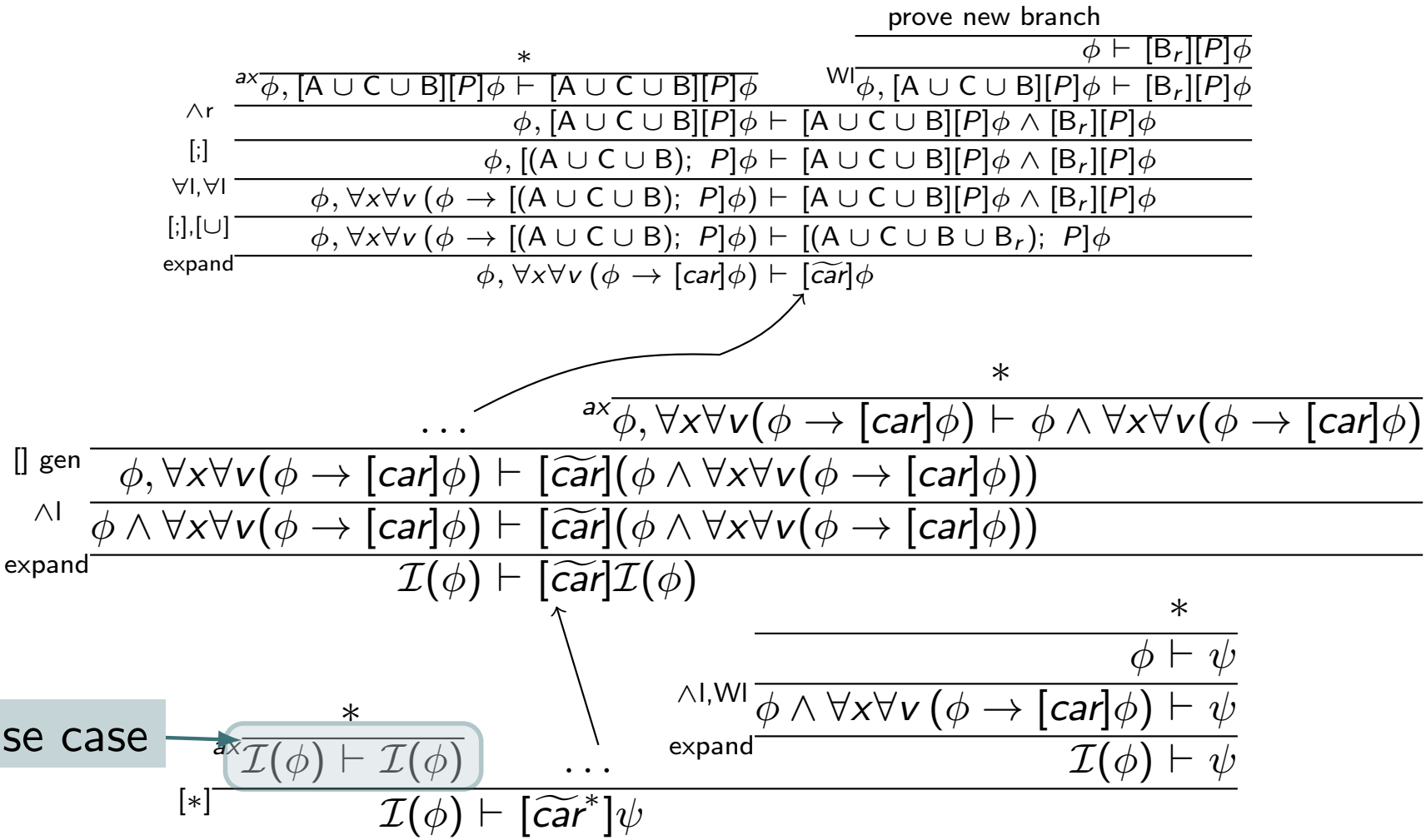
Liveness None

Safety Show that the controller with subsequent complementary dynamics only reaches states that are already reachable with the original dynamics, i. e., show $\left(\langle \alpha; x' = \theta \,\&\, \sim F \rangle \Upsilon_V\right) \rightarrow \left(\langle \alpha; x' = \theta \,\&\, F \rangle \Upsilon_V\right)$

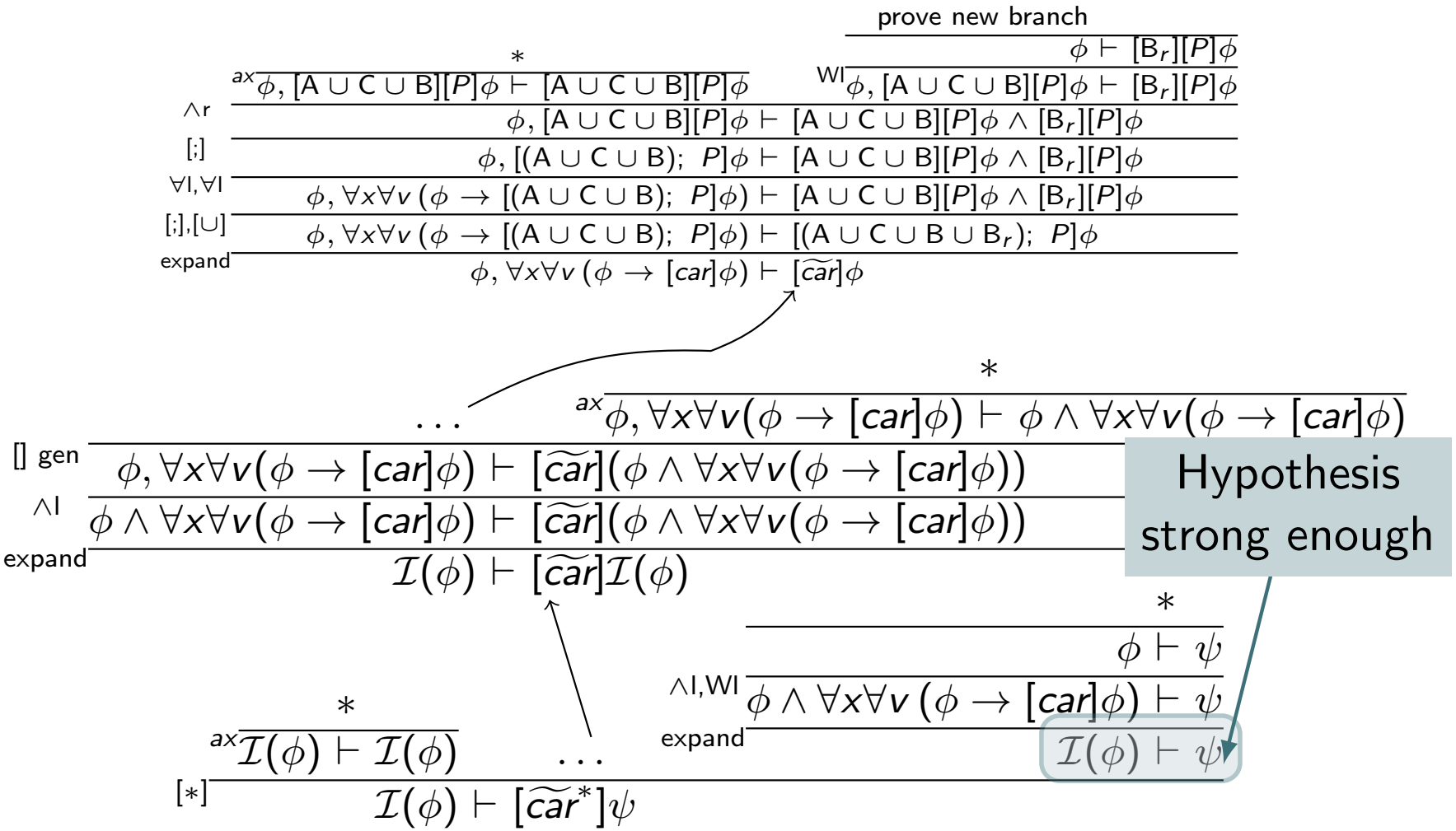
Example: Introduce Moderate Braking



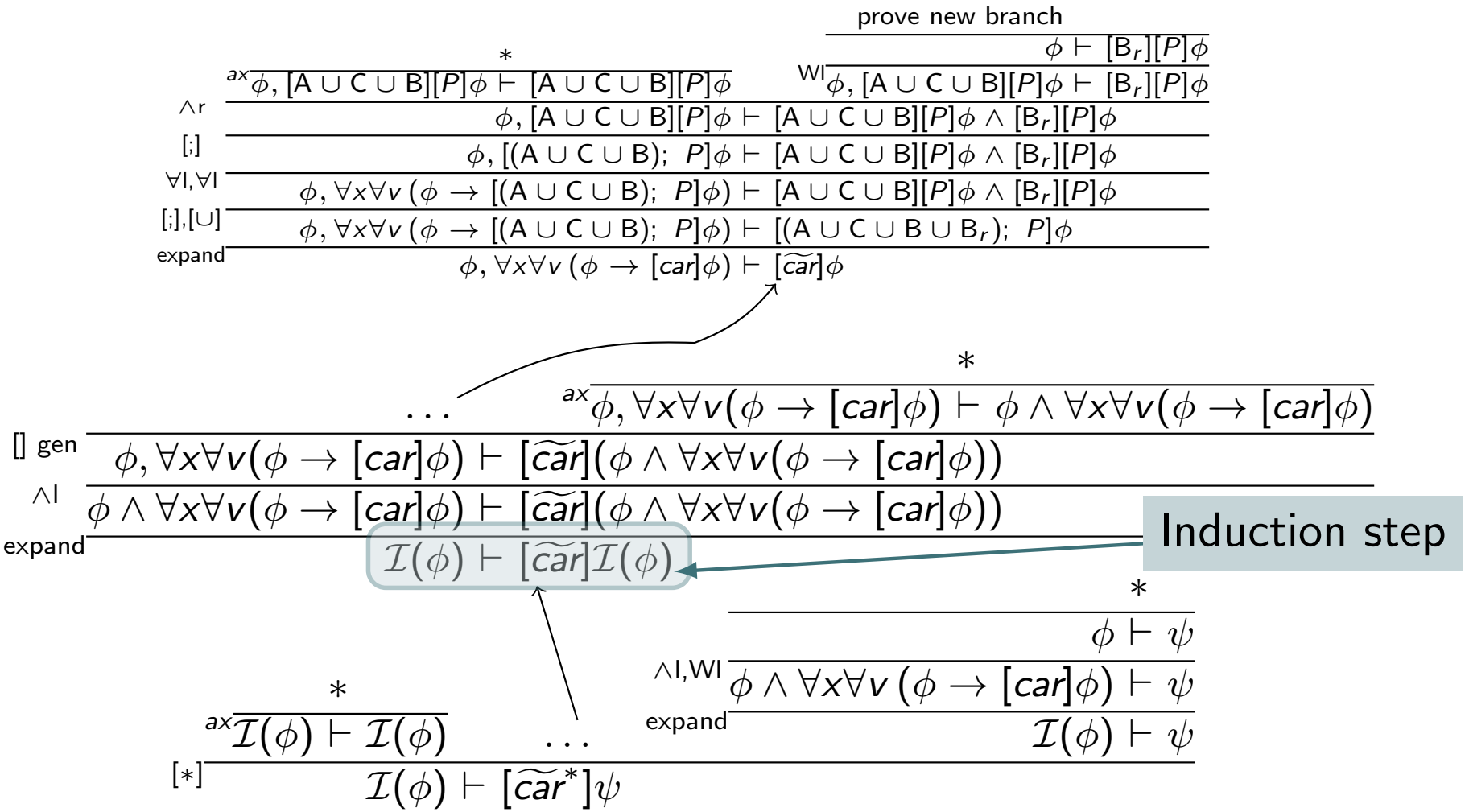
Introduce Moderate Braking: Auxiliary Safety Proof



Introduce Moderate Braking: Auxiliary Safety Proof

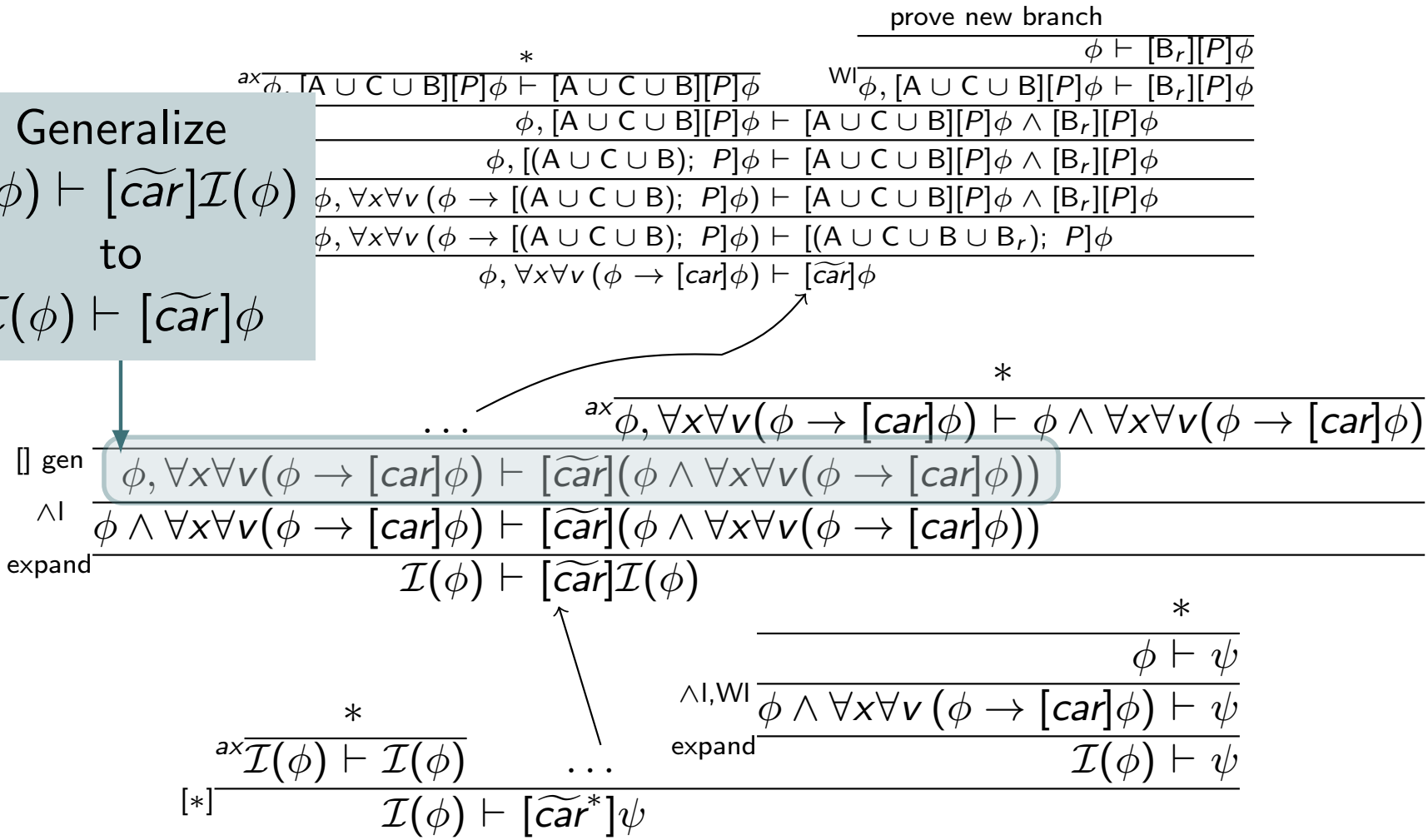


Introduce Moderate Braking: Auxiliary Safety Proof

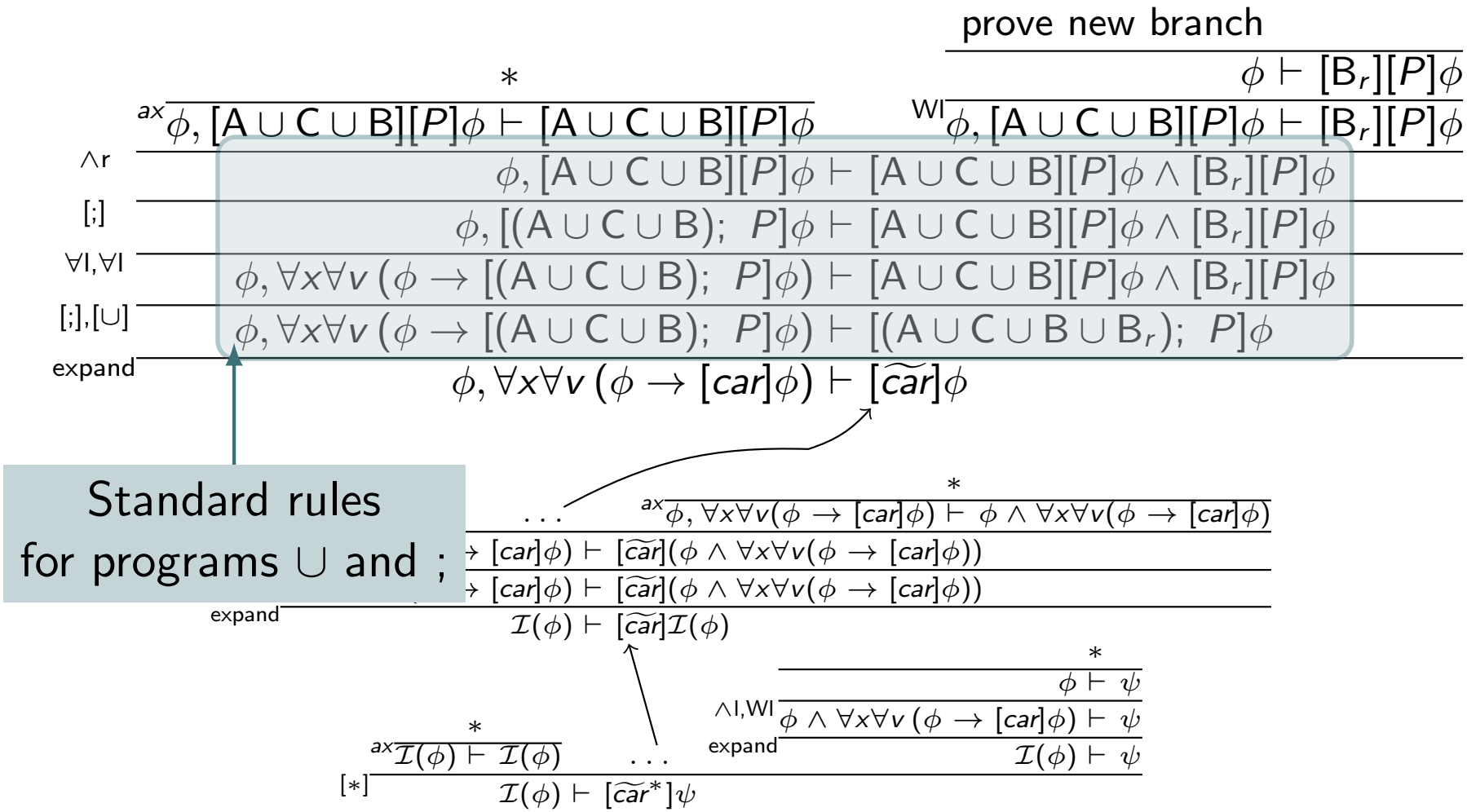


Introduce Moderate Braking: Auxiliary Safety Proof

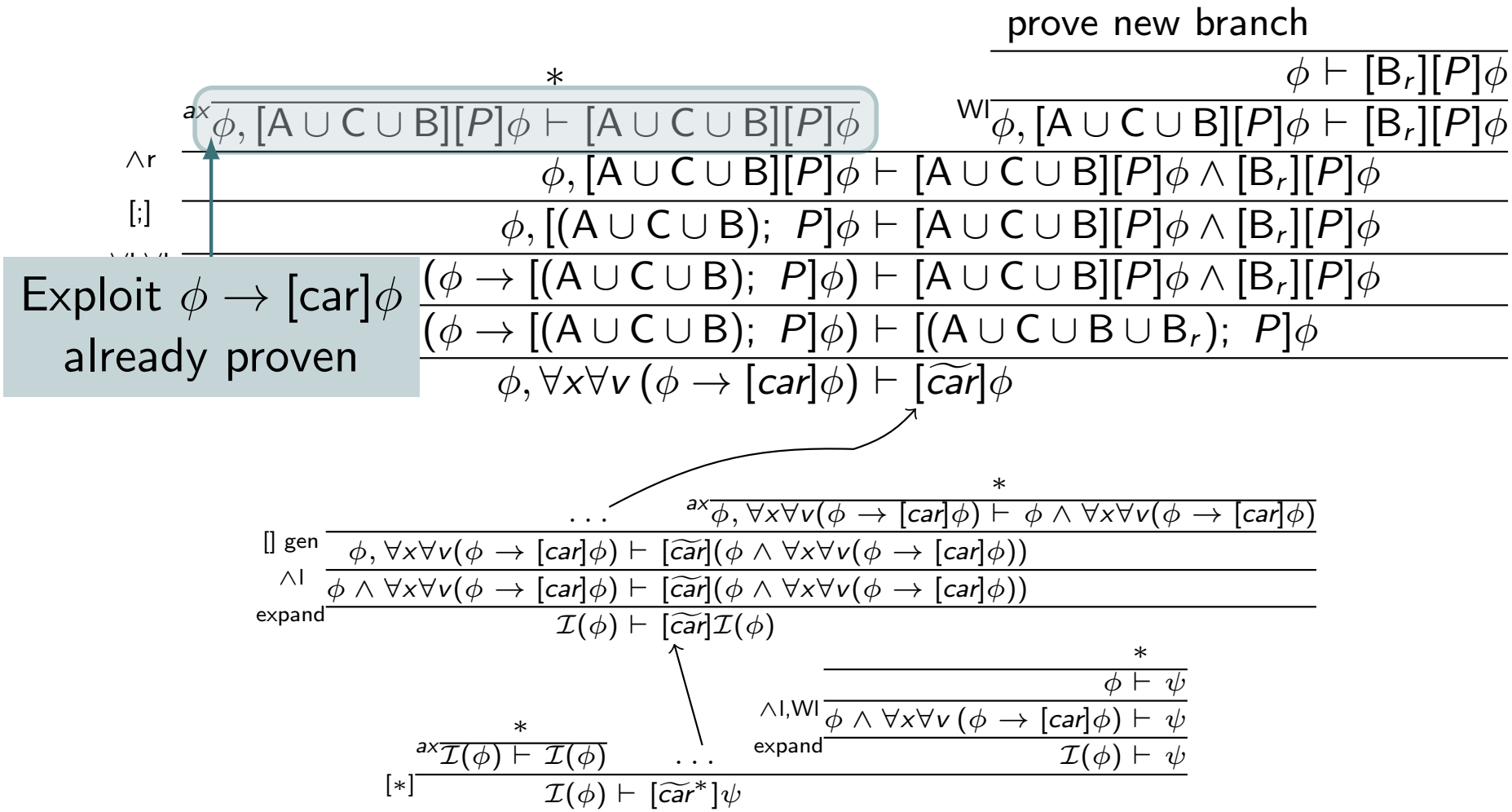
Generalize
$\mathcal{I}(\phi) \vdash [\widetilde{car}]\mathcal{I}(\phi)$
to
$\mathcal{I}(\phi) \vdash [\widetilde{car}]\phi$



Introduce Moderate Braking: Auxiliary Safety Proof



Introduce Moderate Braking: Auxiliary Safety Proof



Introduce Moderate Braking: Auxiliary Safety Proof