

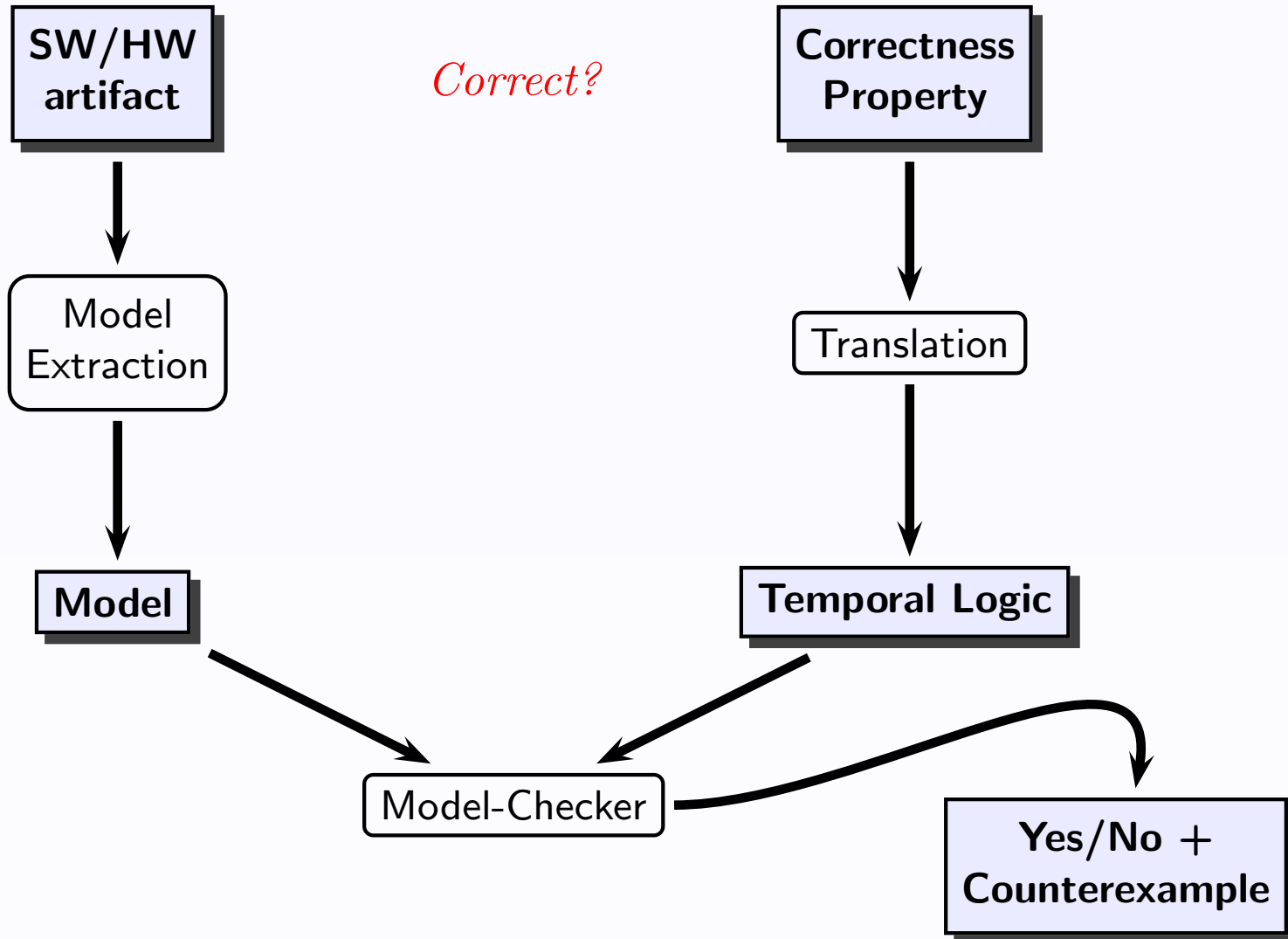
Stuttering Abstraction for Model-Checking

Shiva Nejati **Arie Gurfinkel** **Marsha Chechik**

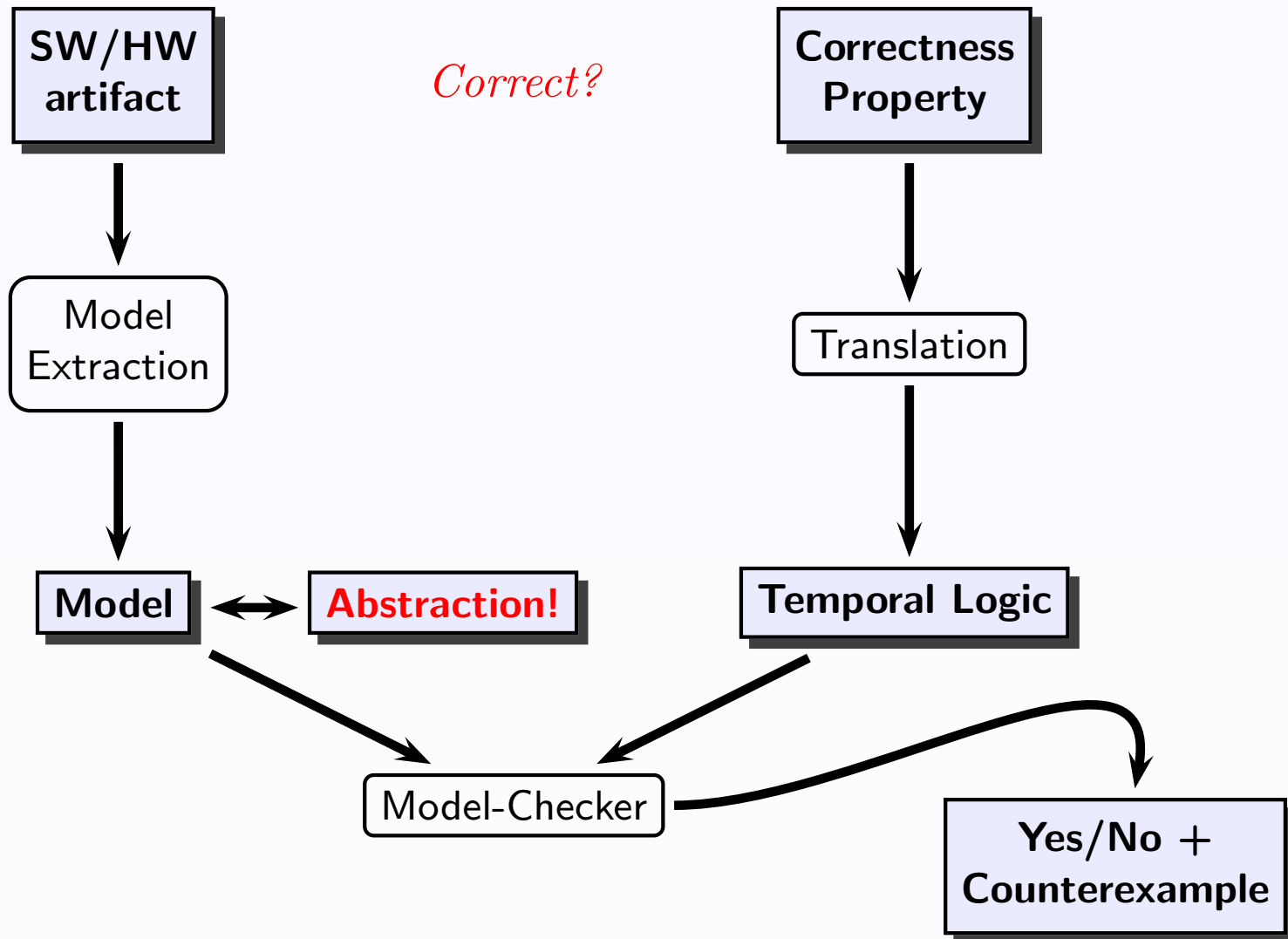
Department of Computer Science
University of Toronto, Canada.

Email: {**shiva**, arie, chechik}@cs.toronto.edu

Overview of Model-Checking



Overview of Model-Checking



Desired Features of Abstraction

→ Sound

↪ If an abstract model A satisfies φ , so does its concretization C

→ Conclusive

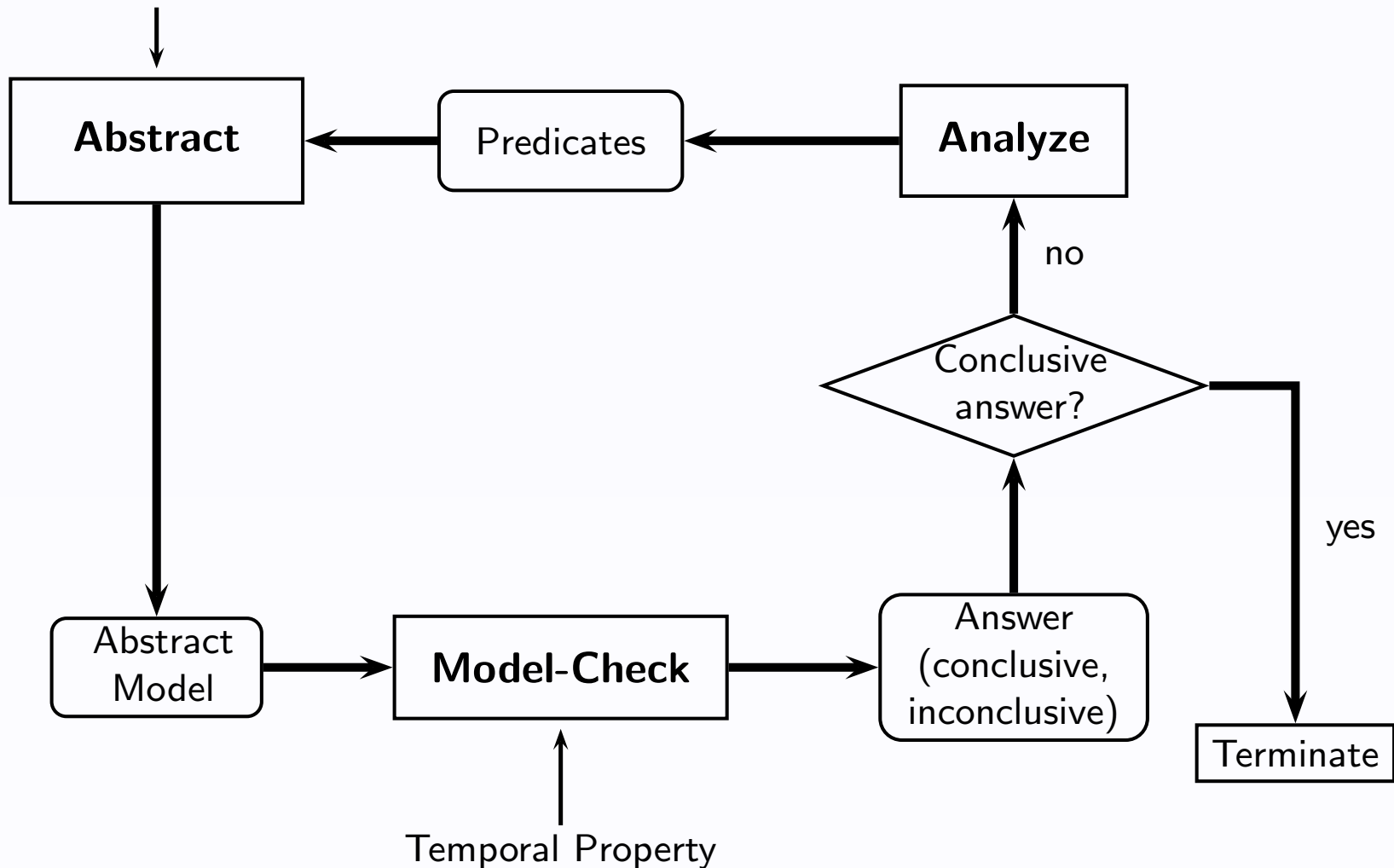
↪ If C satisfies φ , so does some **finite** abstraction A of C

→ Small

↪ An abstraction A should be small enough to be model-checked

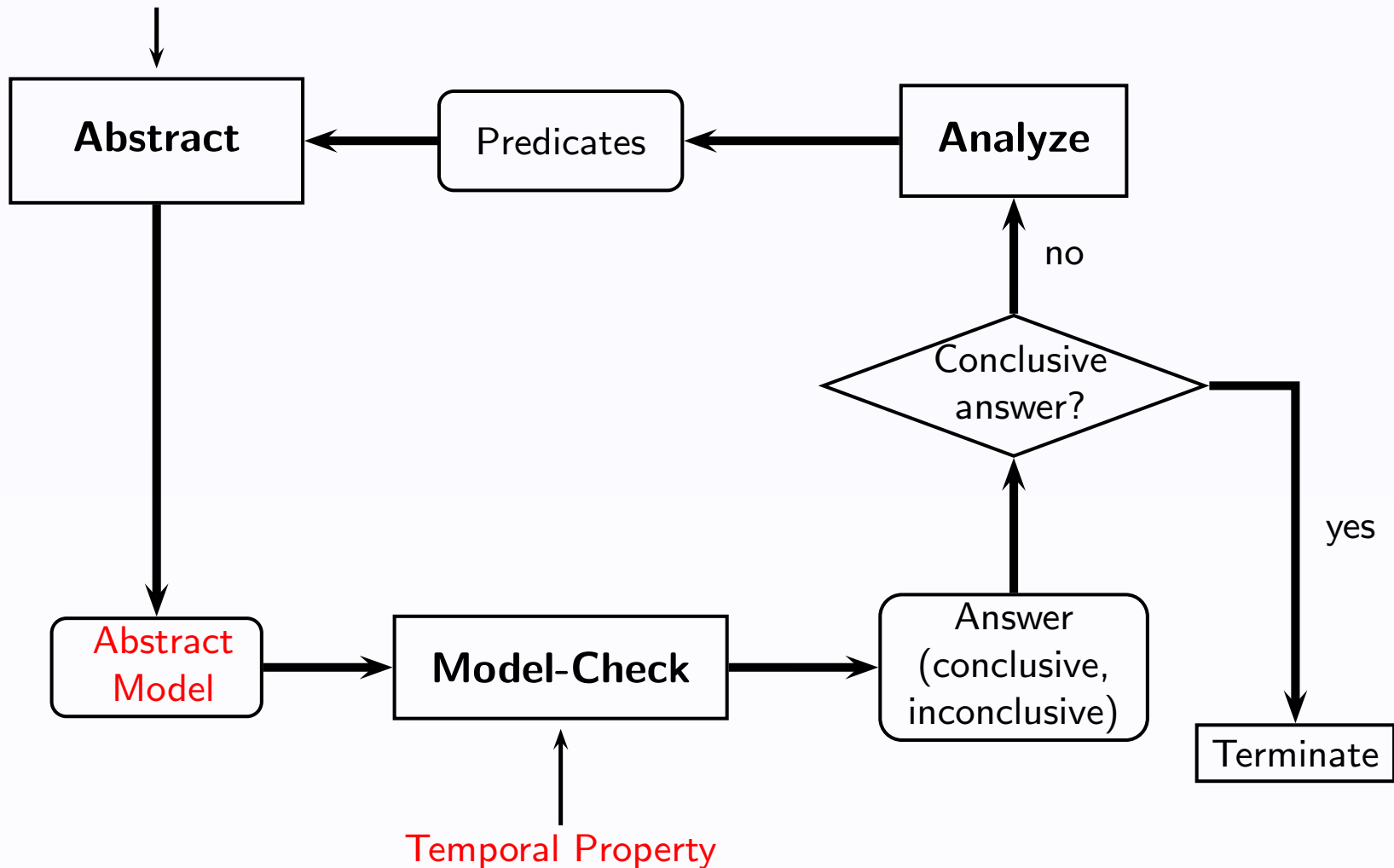
Software Model-Checking: Overview

concrete program + initial predicates



Software Model-Checking: Overview

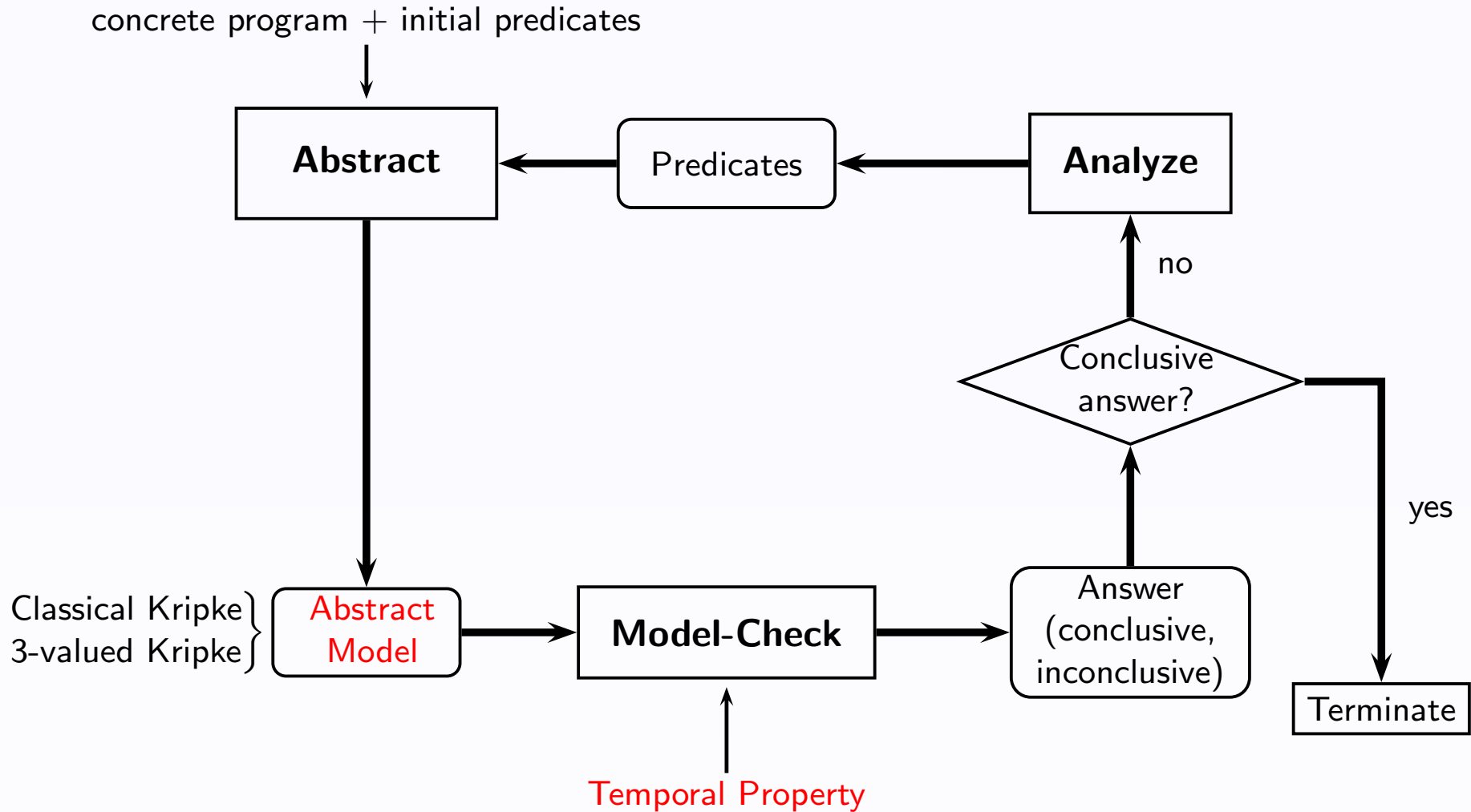
concrete program + initial predicates



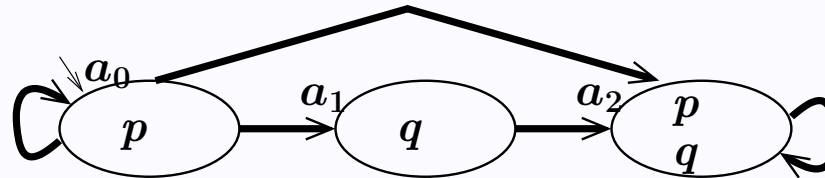
Temporal logic CTL

- ↪ A branching-time temporal logic
- ↪ Allows to quantify over possible futures
 - **Universal Properties**
 - ▮ $AG(p \Rightarrow AFq)$: whenever p holds, q will eventually hold along all paths
 - **Existential Properties**
 - ▮ EFp : p holds in some future state along one path
 - **Mixed Properties**
 - ▮ $AGEFp$: every state can eventually reach a state where p holds

Software Model-Checking: Overview



Classical Kripke Structures



Red arrow pointing right: Abstractions either

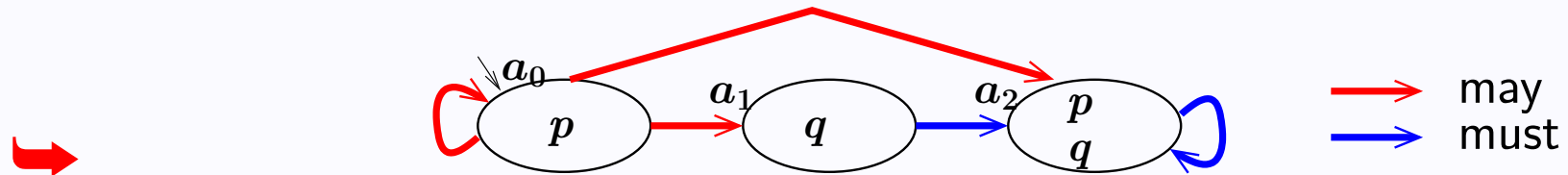
- simulate the concrete models (over-approximations) or are simulated by the concrete models (under-approximations)

Red arrow pointing right: Soundness

- ✓ Universal
- ✓ Existential
- ✗ Mixed

Red arrow pointing right: Can soundness be improved without significantly increasing the size?

3-Valued Kripke Structures



- ➔ Abstractions have two types of transitions: may and must
 - may+must transitions over-approximate and must transitions under-approximate
- ➔ Soundness
 - ✓ Universal
 - ✓ Existential
 - ✓ Mixed
- ➔ What about conclusiveness?

Software MC: Example

Concrete

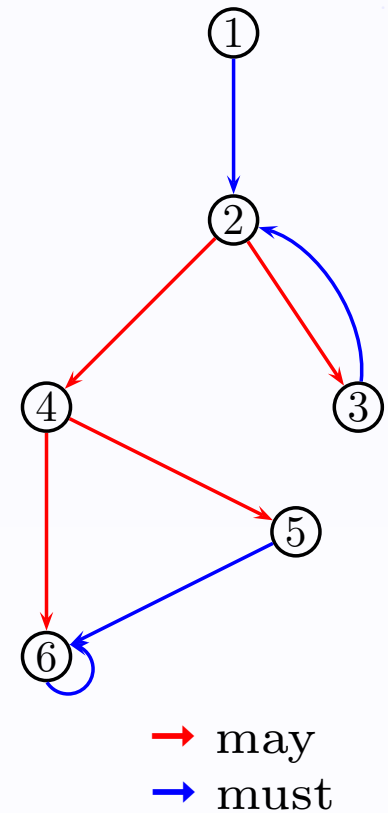
Q: Is Error() reachable?

```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6:}
```

Abstract

Predicate Set = {}

```
void main (void) {
1: ;
2: while (*)
3:   { ; }
4: if (*)
5:   {Error();}
6:}
```



Software MC: Example

Concrete

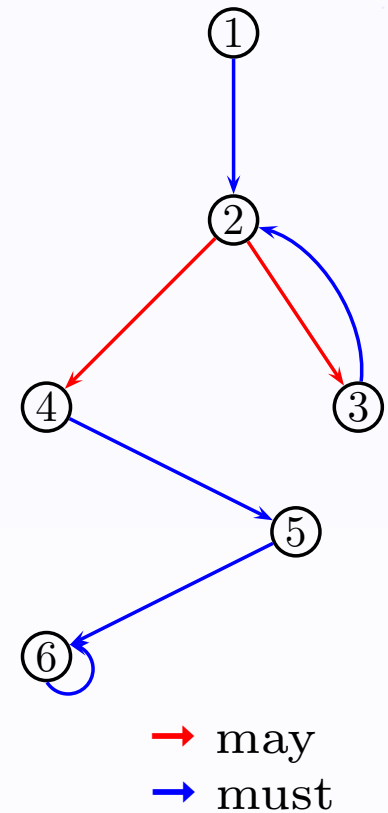
Q: Is Error() reachable?

```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6:}
```

Abstract

Predicate Set = $\{x = 2\}$

```
void main (void) {
1: (x=2) := T;
2: while (*)
3:   {(x=2) := (x=2);}
4: if (x=2)
5:   {Error();}
6:}
```



Software MC: Example

Concrete

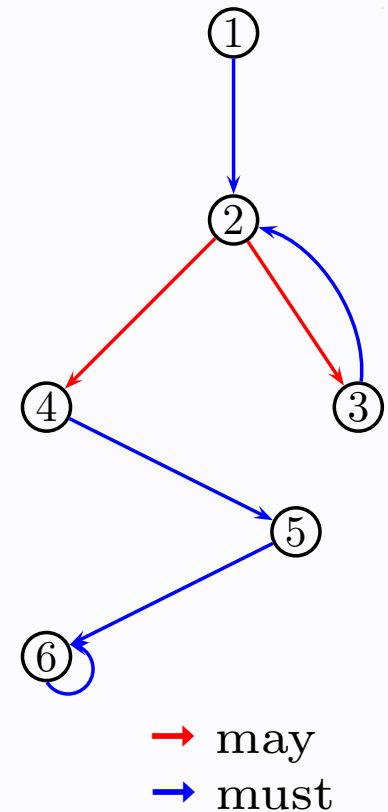
Q: Is Error() reachable?

```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6: }
```

Abstract

Predicate Set = $\{x = 2, y \geq 2\}$

```
void main (void) {
1: (x=2) := T,
   (y>=2) := *;
2: while (y>=2)
3:   {(y>=2) := (y>=2)? * : F;
   (x=2) := (x=2);}
4: if (x=2)
5:   {Error();}
6: }
```



Software MC: Example

Concrete

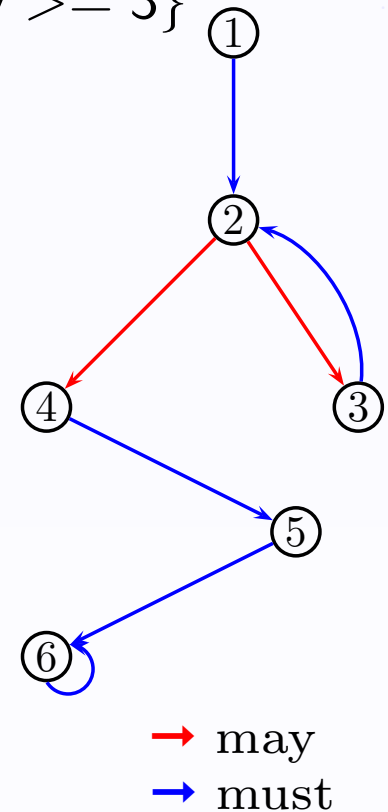
Q: Is Error() reachable?

```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6:}
```

Abstract

Predicate Set = $\{x = 2, y \geq 2, y \geq 3\}$

```
void main (void) {
1: (x=2) := T,
   (y>=2) := *;
   (y>=3) := *;
2: while (y>=2)
3:   {(y>=2) := (y>=3)? T : F;
   (y >=3) := (y>=3)? * : F;
   (x=2) := (x=2);}
4: if (x=2)
5:   {Error();}
6:}
```



Software MC: Example

Concrete

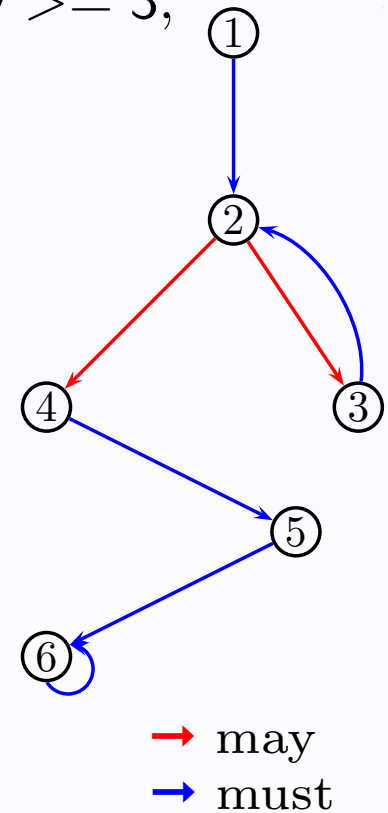
Q: Is Error() reachable?

```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6:}
```

Abstract

Predicate Set = $\{x = 2, y \geq 2, y \geq 3, \dots, ?\}$

```
void main (void) {
1: (x=2) := T,
   (y>=2) := *;
   (y>=3) := *;
2: while (y>=2)
3:   {(y>=2) := (y>=3)? T : F;
   (y >=3) := (y>=3)? * : F;
   (x=2) := (x=2);}
4: if (x=2)
5:   {Error();}
6:}
```



Our Goal

- **Characterizing abstract models that are**
 - ↳ **conclusive for arbitrary program statements**
 - **e.g. loops and recursive functions**
 - ↳ **small and compatible with current model-checking algorithms**

Software MC: Problem

Concrete

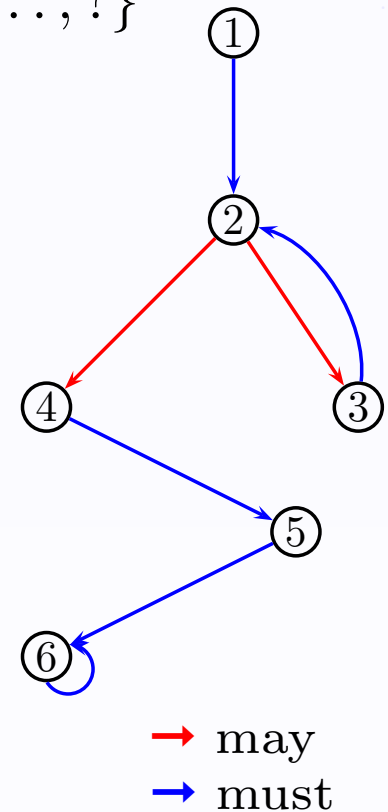
Q: Is Error() reachable?

```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6:}
```

Abstract

Predicate Set = $\{x = 2, y \geq 2, \dots, ?\}$

```
void main (void) {
1: (x=2) := T,
   (y>=2) := *;
2: while (y>=2)
3:   {(y>=2) := (y>=2)? * : F;
   (x=2) := (x=2);}
4: if (x=2)
5:   {Error();}
6:}
```



Our Contribution

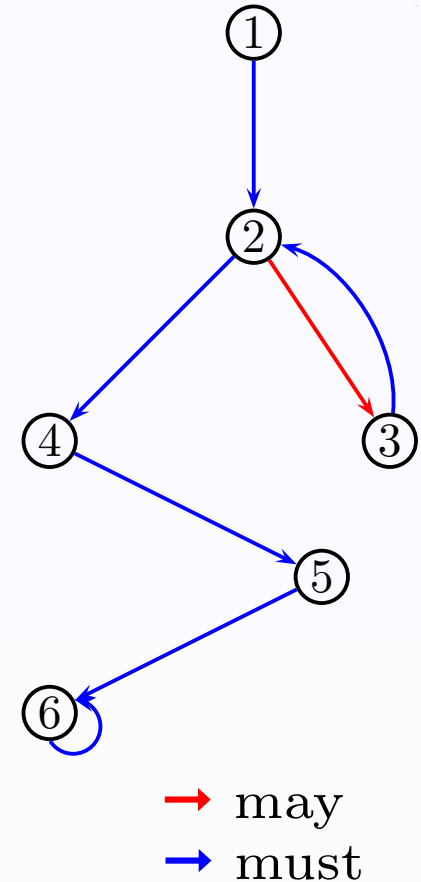
→ New Abstraction : Stuttering Abstraction

- ➔ Models: 3-valued Kripke structures
- ➔ Soundness: CTL without next-time operators

✗ **A property with next:** $EX(y \geq 2)$
 $y \geq 2$ holds in the next state

✓ **A property without next:** $EF(y \geq 2)$
 $y \geq 2$ holds in some future state

- ➔ Conclusiveness: loops + recursion



The Rest . . .

- ↳ Relationships between abstract and concrete models
- ↳ Properties for which abstractions are sound
- ↳ Constructing abstract models

The Rest . . .

- ↳ Relationships between abstract and concrete models
- ↳ Properties for which abstractions are sound
- ↳ Constructing abstract models

Relations: Refinement

Simulation [Milner71]

Over Kripke structures

- A simulates C if
 - ▮ C matches all A 's moves

Preserves

- ✓ Universal
- ✓ Existential
- ✗ Mixed

Relations: Refinement

Simulation [Milner71]

Over Kripke structures

- A simulates C if
 - ▮▮▮ C matches all A 's moves

Preserves

- ✓ Universal
- ✓ Existential
- ✗ Mixed

Refinement [Larsen88]

Over 3-valued Kripke structures

- C refines A if
 - ▮▮▮ A_{must} simulates C_{must}
must trans. are preserved
 - ▮▮▮ C_{may} simulates A_{may}
may trans. go away or
are converted to must trans.

Preserves

- ✓ Universal
- ✓ Existential
- ✓ Mixed

Relations: Stuttering Refinement

Stuttering Simulation

[Browne-Clarke-Grumberg88]

Over Kripke structures

- A **stut-simulates** C if
 - ▮ **finite** (infinite) sequences of states in C match **finite** (infinite) sequences of states in A

Preserves

- ✓ **Universal** (without next)
- ✓ **Existential** (without next)
- ✗ **Mixed** (without next)

Relations: Stuttering Refinement

Stuttering Simulation

[Browne-Clarke-Grumberg88]

Over Kripke structures

- A **stut-simulates** C if
 - ▮ **finite** (infinite) sequences of states in C match **finite** (infinite) sequences of states in A

Preserves

- ✓ Universal (without next)
- ✓ Existential (without next)
- ✗ Mixed (without next)

Stuttering Refinement

[Nejati-Gurfinkel-Chechik05]

Over 3-valued Kripke structures

- C **stut-refines** A if
 - ▮ A_{must} stut-simulates C_{must}
 - ▮ C_{may} stut-simulates A_{may}

Preserves

- ✓ Universal (without next)
- ✓ Existential (without next)
- ✓ Mixed (without next)

Relations: Divergence-blind Ref.

Divergence-blind Simulation

[deNicola-Vaandrager95]

Over Kripke structures

- A **db-simulates** C if
 - ▮ **finite** sequences of states in C match **finite** or **infinite** sequences of states in A

Preserves

- ✓ **Universal** (w. finite witnesses)
- ✓ **Existential** (w. finite witnesses)
- ✗ **Mixed** (w. finite witnesses)

Relations: Divergence-blind Ref.

Divergence-blind Simulation

[deNicola-Vaandrager95]

Over Kripke structures

- A **db-simulates** C if
 - ▮ **finite** sequences of states in C match **finite** or **infinite** sequences of states in A

Preserves

- ✓ Universal (w. finite witnesses)
- ✓ Existential (w. finite witnesses)
- ✗ Mixed (w. finite witnesses)

Divergence-blind Refinement

[Nejati-Gurfinkel-Chechik05]

Over 3-valued Kripke structures

- C **db-refines** A if
 - ▮ A_{must} db-simulates C_{must}
 - ▮ C_{may} db-simulates A_{may}

Preserves

- ✓ Universal (w. finite witnesses)
- ✓ Existential (w. finite witnesses)
- ✓ Mixed (w. finite witnesses)

Refinement vs. Stuttering Refinement

Concrete

Q: Is Error() reachable?

```
void main (int y) {  
1: int x = 2;  
2: while (y >= 2)  
3:   {y = y - 1;}  
4: if (x == 2)  
5:   {Error();}  
6:}
```

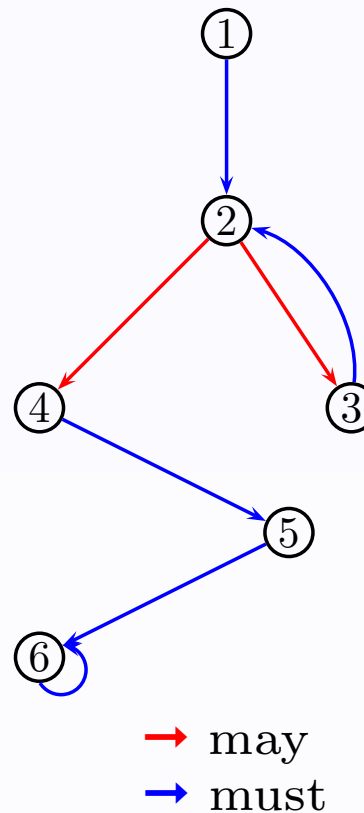
Refinement vs. Stuttering Refinement

Concrete

Q: Is Error() reachable?

```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6: }
```

Refinement Abstraction



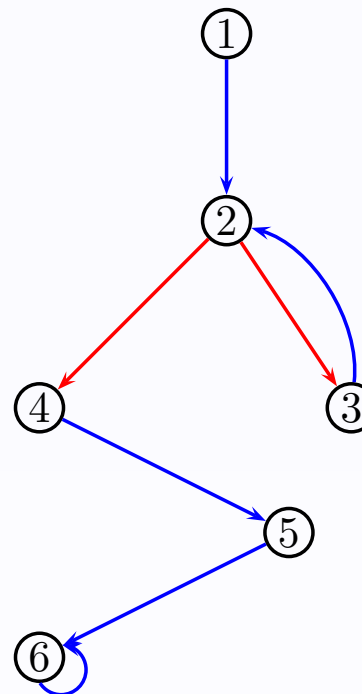
Refinement vs. Stuttering Refinement

Concrete

Q: Is Error() reachable?

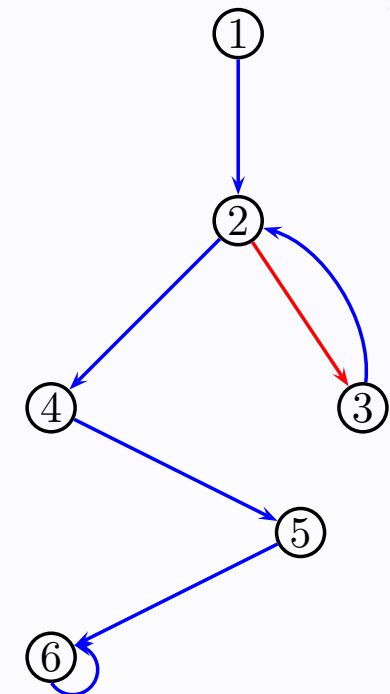
```
void main (int y) {
1: int x = 2;
2: while (y >= 2)
3:   {y = y - 1;}
4: if (x == 2)
5:   {Error();}
6: }
```

Refinement Abstraction



→ may
→ must

Stuttering Abstraction



→ may
→ must

The Rest . . .

- ↳ Relationships between abstract and concrete models
- ↳ Properties for which abstractions are sound
- ↳ Constructing abstract models

CTL and Its Fragments

→ CTL operators

- ↪ Adequate set: p , \neg , \vee , EX , EU , and EG
 - $EX\varphi$: φ is true in some next state
 - $E[\varphi_1 U \varphi_2]$: along some path φ_1 holds until φ_2 holds
 - $EG\varphi$: along some path φ holds in every state
- ↪ Universal properties: $AX\varphi$, $AG\varphi$, $A[\varphi_1 U \varphi_2]$

CTL and Its Fragments

→ CTL operators

→ Adequate set: p , \neg , \vee , EX , EU , and EG

➤ $EX\varphi$: φ is true in some next state

➤ $E[\varphi_1 U \varphi_2]$: along some path φ_1 holds until φ_2 holds

➤ $EG\varphi$: along some path φ holds in every state

→ Universal properties: $AX\varphi$, $AG\varphi$, $A[\varphi_1 U \varphi_2]$

→ CTL Fragments

	p	\vee	EX	EU	EG	\neg			p	\vee	EX	EU	EG
CTL	✓	✓	✓	✓	✓	✓		ECTL	✓	✓	✓	✓	✓
CTL _{-X}	✓	✓	✗	✓	✓	✓		ECTL _{-X}	✓	✓	✗	✓	✓
CTL _U	✓	✓	✗	✓	✗	✓		ECTL _U	✓	✓	✗	✓	✗

Soundness of Abstractions

- ➔ Relation ρ preserves a logic L (x)
 ρ relates A and $C \Rightarrow A$ is sound for L
- ➔ Relation ρ is characterized by a logic L (*)
 ρ relates A and $C \Leftrightarrow A$ is sound for L

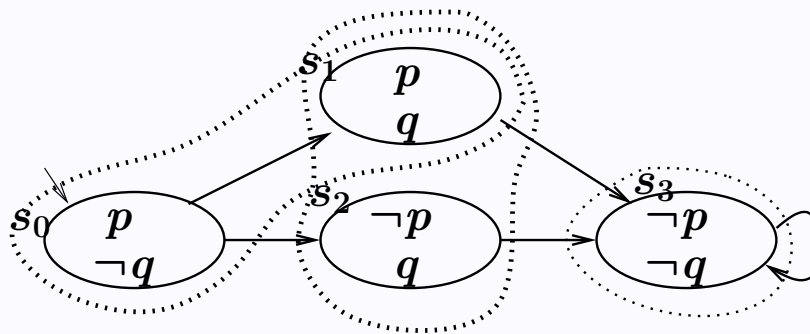
	Subset of ECTL		
Simulation relations	ECTL _U	ECTL _{-X}	ECTL
Classical	x	x	*
Stuttering	x	*	
Divergence-blind	*		
	Subset of CTL		
Refinement relations	CTL _U	CTL _{-X}	CTL
Refinement	x	x	*
Stuttering	x	*	
Divergence-blind	*		

The Rest . . .

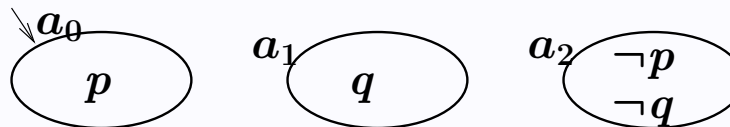
- ↳ Relationships between abstractions and concrete models
- ↳ Properties for which abstractions are sound
- ↳ Constructing abstract models

Constructing Abstractions

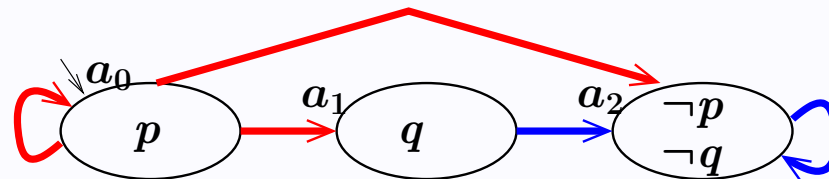
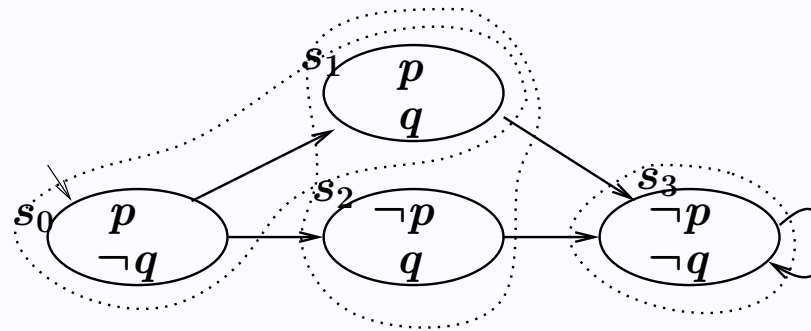
Concrete (\mathbf{C})



Abstract (\mathbf{A})



Constructing Refinement Abstractions



$a \rightarrow a'$ (must):

iff $\forall s$ related to a , $\exists s'$ s.t.

- s' is related to a' , and
- $s \rightarrow s'$

$R^{\forall\exists}$ condition [Dams96]

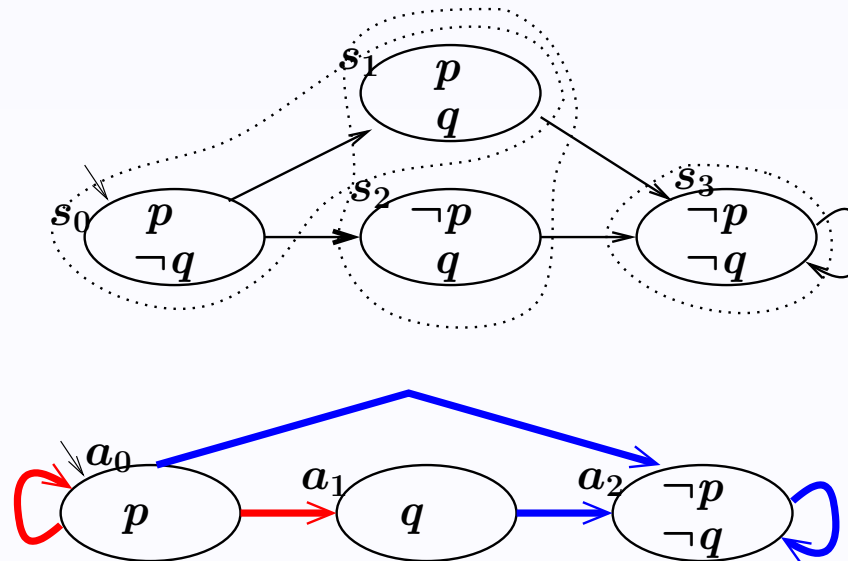
$a \rightarrow a'$ (may):

iff $\exists s$ related to a , and $\exists s'$ s.t.

- s' is related to a' , and
- $s \rightarrow s'$

$R^{\exists\exists}$ condition [Dams96]

Constructing Stuttering Abstractions



$a \rightarrow a'$ (must):

iff $\forall s$ related to a , $\exists s_1, \dots, s_n$ s.t.

- s_i is related to a for $i < n$ and s_n is related to a' , and
- $s \rightarrow s_1 \rightarrow \dots \rightarrow s_n$

Weaker $R^{\forall\exists}$ condition

Computation of Stuttering Abstraction

→ Problem

- ↪ Requires computing reachability, not locally computable

→ Possible solutions

- ↪ Distance-bounded reachability instead of reachability
 - reachability up to a certain bound k
 - ▮ for $k = 1$, we obtain refinement
- ↪ Using deductive techniques
 - not automated, but allows integration of model-checking and theorem-proving

Related Work

→ Abstractions as automata

[Namjoshi04,Dams-Namjoshi05]

- ↪ Automata-based structures are conclusive for arbitrary temporal properties, but
 - abstract domains may become significantly larger
 - model-checking and construction of such abstractions still needs to be studied

Summary

→ Stuttering abstraction

- ↳ keeps abstract models small and simple
- ↳ relaxes the relation between abstract and concrete models
 - the abstraction becomes sound for fewer properties, but is more conclusive for those properties
- ↳ provides abstract models for some classes of properties
 - . . . for which conventional refinement/abstraction does not converge in finite time

Future Work

- ↪ Characterizing the properties for which stuttering abstraction is conclusive
- ↪ Effective techniques for computing stuttering abstraction
- ↪ A methodology for identifying applicability of stuttering abstraction in software model-checking

Thank You!

Questions?