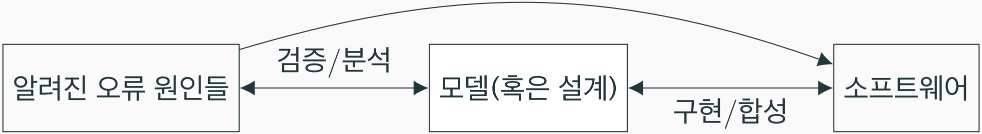
패턴 기반 모델검증을 위한 오류 탐색 전략

배경민

2023년 2월 1일

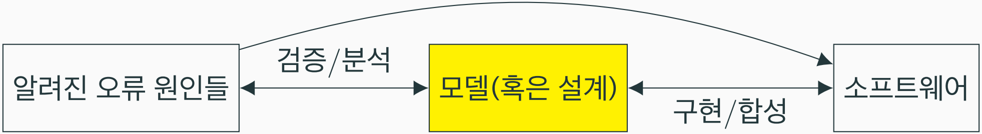
POSTECH 컴퓨터공학과

그룹3 연구목표: 소프트웨어재난 재발방지



- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 연구내용2: 모델(혹은 설계) 단계에서 소프트웨어 재난의 원인을 분석
- 연구내용3: 모델 합성, 요구사항 추론, 오류패턴 기반 모델검증 등 신기술 개발

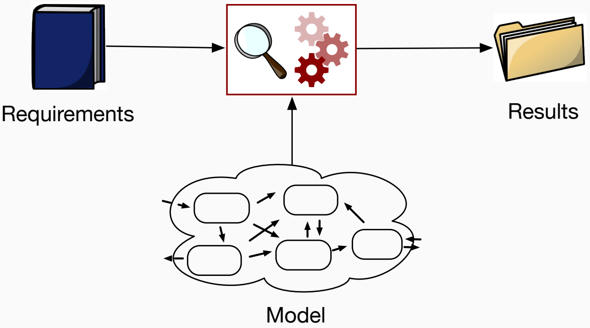
그룹3 연구목표: 소프트웨어재난 재발방지



- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 연구내용2: 모델(혹은 설계) 단계에서 소프트웨어 재난의 원인을 분석
- 연구내용3: 모델 합성, 요구사항 추론, 오류패턴 기반 모델검증 등 신기술 개발

모델검증 (Model Checking)

- 시스템의 오류를 자동으로 찾는 기술
  - 소프트웨어/하드웨어 디자인, 프로토콜 디자인, 소스 코드, …
- 특징
  - 시스템의 모든 가능한 상태를 확인하여 "오류 없음" 증명 가능
- 문제점
  - 상태공간폭발, …



POSTECH 정형명세 및 모델검증 연구 (1)

- 접근방법: 논리 기반 모델 검증

Model		Logic System		Verification
시스템 명세		수학적 모델		
M	$\Longrightarrow$	$\mathcal{R}_M$		모델검증
			$\Longrightarrow$	
성질 명세		논리식		알고리즘
$spec$	$\Longrightarrow$	φ_{spec}		

- 대상 성질/오류 및 시스템에 최적화된 모델링 및 정형명세 기술 연구
- 논리 시스템의 알고리즘 및 최적화 기법 연구: Rewriting logic 및 SMT

POSTECH 정형명세 및 모델검증 연구 (2)

- 패턴을 활용한 모델검증 알고리즘 연구
  - 시스템 상태 및 오류의 패턴에 기반한 모델검증
  - 효과적인 상태공간 탐색 전략 및 추상화 기법 연구

POSTECH 정형명세 및 모델검증 연구 (2)

- 패턴을 활용한 모델검증 알고리즘 연구
  - 시스템 상태 및 오류의 패턴에 기반한 모델검증
  - 효과적인 상태공간 탐색 전략 및 추상화 기법 연구
- 다양한 도메인에 대하여 논리 기반 모델 검증 적용
  - TLS 소프트웨어의 정형명세 및 모델검증 (포스터: 이재훈)
  - Promela 언어의 Rewriting 기반 정형명세 및 모델검증 (포스터: 손병호)

POSTECH 정형명세 및 모델검증 연구 (2)

- 패턴을 활용한 모델검증 알고리즘 연구
  - 시스템 상태 및 오류의 패턴에 기반한 모델검증
  - 효과적인 상태공간 탐색 전략 및 추상화 기법 연구
- 다양한 도메인에 대하여 논리 기반 모델 검증 적용
  - TLS 소프트웨어의 정형명세 및 모델검증 (포스터: 이재훈)
  - Promela 언어의 Rewriting 기반 정형명세 및 모델검증 (포스터: 손병호)
- 사이버물리시스템 (Cyber-physical systems) 모델검증 연구
  - Signal Temporal Logic 모델 검증 연구 (포스터: 이지아)
  - 상태공간 축소기법 (포스터: 이재서)

POSTECH 정형명세 및 모델검증 연구 (2)

- 패턴을 활용한 모델검증 알고리즘 연구
  - 시스템 상태 및 오류의 패턴에 기반한 모델검증
  - 효과적인 상태공간 탐색 전략 및 추상화 기법 연구
- 다양한 도메인에 대하여 논리 기반 모델 검증 적용
  - TLS 소프트웨어의 정형명세 및 모델검증 (포스터: 이재훈)
  - Promela 언어의 Rewriting 기반 정형명세 및 모델검증 (포스터: 손병호)
- 사이버물리시스템 (Cyber-physical systems) 모델검증 연구
  - Signal Temporal Logic 모델 검증 연구 (포스터: 이지아)
  - 상태공간 축소기법 (포스터: 이재서)
- Deep neural network (DNN) 검증 연구
  - DNN 검증을 위한 요약 해석 기법 연구 (포스터: 연주은)
  - DNN 검증 시 발생하는 conflict 정보를 이용한 성능 향상 기법 연구

진행연구 소개: 패턴을 활용한 모델검증 및 탐색 전략

연구개요: 패턴을 활용한 모델검증

- 목적: 모델 검증 시 패턴에 기반한 상태공간 탐색
  - 오류 패턴 등을 활용하여 (알려진) 오류를 효율적으로 탐색
  - 오류와 관계가 적은 상태공간을 효과적으로 요약

연구개요: 패턴을 활용한 모델검증

- 목적: 모델 검증 시 패턴에 기반한 상태공간 탐색
    - 오류 패턴 등을 활용하여 (알려진) 오류를 효율적으로 탐색
    - 오류와 관계가 적은 상태공간을 효과적으로 요약
- 필요 기술
    - 패턴의 정의 및 패턴 기반 실행 방법
    - 패턴 공간의 탐색 및 요약

연구개요: 패턴을 활용한 모델검증

- 목적: 모델 검증 시 패턴에 기반한 상태공간 탐색
  - 오류 패턴 등을 활용하여 (알려진) 오류를 효율적으로 탐색
  - 오류와 관계가 적은 상태공간을 효과적으로 요약
- 필요 기술
  - 패턴의 정의 및 패턴 기반 실행 방법
  - 패턴 공간의 탐색 및 요약
- 연구 방법
  - Rewriting logic: 높은 표현력을 가진 명세 언어로 다양한 모델링 언어의 의미 정의 가능
  - Rewriting logic으로 명세된 시스템의 패턴 기반 모델검증 연구

Rewriting Logic Specification

- State

  term t (algebraic data type)

- Transition

  rewrite rule $t \longrightarrow t'$ (patterns t and t')

- Example
  - By rule $f(N) \longrightarrow f(s(N))$,

Rewriting Logic Specification

- State

  term t (algebraic data type)

- Transition

  rewrite rule $t \longrightarrow t'$ (patterns t and t')

- Example
  - By rule $f(N) \longrightarrow f(s(N))$, term $f(s(0))$ is rewritten to

Rewriting Logic Specification

- State

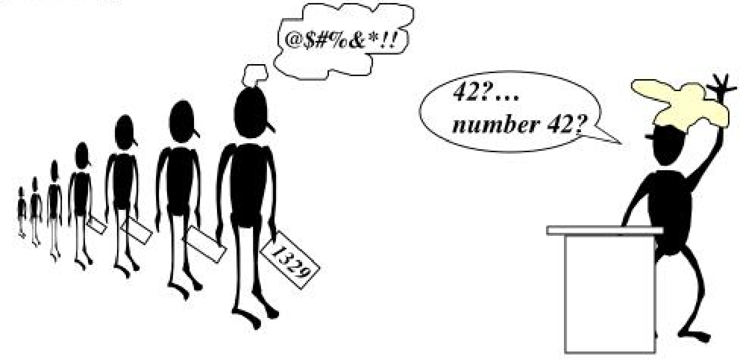
  term t (algebraic data type)

- Transition

  rewrite rule $t \longrightarrow t'$ (patterns t and t')

- Example
  - By rule $f(N) \longrightarrow f(s(N))$, term $f(s(0))$ is rewritten to $f(s(s(0)))$

Example: Lamport's Bakery Algorithm



- Each process receives a ticket number to enter the critical section.
- Process with the smallest ticket number enters the critical section.

Example: Lamport's Bakery Algorithm

- Each state with N processes:

$$n \; ; \; m \; ; \; [i_1, d_1] \ldots [i_N, d_N]$$

  - n: the current number in the bakery's number dispenser
  - m: the number currently served
  - $[i_l, d_l]$: process with id d_l in status $d_l \in \{\texttt{idle}, \texttt{wait}(ticket), \texttt{crit}(ticket)\}$

Example: Lamport's Bakery Algorithm

- Each state with N processes:

$$n \;;\; m \;;\; [i_1, d_1] \ldots [i_N, d_N]$$

  - n: the current number in the bakery's number dispenser
  - m: the number currently served
  - $[i_l, d_l]$: process with id d_l in status $d_l \in \{\texttt{idle}, \texttt{wait}(\mathit{ticket}), \texttt{crit}(\mathit{ticket})\}$

- Rewrite rules (in the Maude syntax)

```
rl [wake]: N ; M ; [K,    idle] PS  =>  s N ; M ; [K,wait(N)] PS .
rl [crit]: N ; M ; [K,wait(M)] PS  =>  N ;   M ; [K,crit(M)] PS .
rl [exit]: N ; M ; [K,crit(M)] PS  =>  N ; s M ; [K,    idle] PS .
```

  - Number represented as multisets: $0 = 0$, $1 = s$, $2 = s\,s$, $\cdots$
  - Variables: `N`, `M`, `K` for numbers, and `PS` for process sets

Example: Lamport's Bakery Algorithm

- Rewrite sequences with one process

$$
\begin{tikzcd}
\texttt{0 ; 0 ; [0,idle]} \arrow[d, "wake"] & \texttt{s ; s ; [0,idle]} \arrow[d, "wake"] & \texttt{s s ; s s ; [0,idle]} \arrow[d, "wake"] \\
\texttt{s ; 0 ; [0,wait(0)]} \arrow[d, "crit"] & \texttt{s s ; s ; [0,wait(s)]} \arrow[d, "crit"] & \texttt{s s s ; s s ; [0,wait(s s)]} \arrow[d, "crit"] \\
\texttt{s ; 0 ; [0,crit(0)]} \arrow[uur, "exit"] & \texttt{s s ; s ; [0,crit(s)]} \arrow[uur, "exit"] & \cdots
\end{tikzcd}
$$

Example: Lamport's Bakery Algorithm

- Rewrite sequences with one process

$$
\begin{tikzcd}
\texttt{0 ; 0 ; [0,idle]} \arrow[d, "wake"] & \texttt{s ; s ; [0,idle]} \arrow[d, "wake"] & \texttt{s s ; s s ; [0,idle]} \arrow[d, "wake"] \\
\texttt{s ; 0 ; [0,wait(0)]} \arrow[d, "crit"] & \texttt{s s ; s ; [0,wait(s)]} \arrow[d, "crit"] & \texttt{s s s ; s s ; [0,wait(s s)]} \arrow[d, "crit"] \\
\texttt{s ; 0 ; [0,crit(0)]} \arrow[uur, dashed, "exit"] & \texttt{s s ; s ; [0,crit(s)]} \arrow[uur, dashed, "exit"] & \cdots
\end{tikzcd}
$$

- Infinite-state system: unbounded counters n and m

Symbolic Representation Using Logical Terms

- Symbolic state

  pattern t with logical variables

Symbolic Representation Using Logical Terms

- Symbolic state

  pattern t with logical variables

- Example
  - `N ; N ; PS`

Symbolic Representation Using Logical Terms

- Symbolic state

  pattern t with logical variables

- Example
  - `N ; N ; PS`
- Express a set of (potentially infinitely many) concrete states
  - $0\,;\,0\,;\,[0, \mathit{idle}],\quad s\,;\,s\,;\,[0, \mathit{idle}]\,[s, \mathit{idle}],\quad s\,s\,;\,s\,s\,;\,[0, \mathit{idle}]\,[s\,s, \mathit{wait}(0)],\quad \cdots$

Computing Symbolic State Space

- Narrowing $\rightsquigarrow$
  - $t \rightsquigarrow t'$ if an instance of t can be rewritten to t' by some rule $l \longrightarrow r$
  - defines symbolic transitions between logical terms

Computing Symbolic State Space

- Narrowing $\rightsquigarrow$
  - $t \rightsquigarrow t'$ if an instance of t can be rewritten to t' by some rule $l \longrightarrow r$
  - defines symbolic transitions between logical terms
- Example

```
rl [wake]: N ; M ; [K,idle] PS => s N ; M ; [K,wait(N)] PS .
```

  - no rewriting from `N ; N ; PS`

Computing Symbolic State Space

- Narrowing $\rightsquigarrow$
  - $t \rightsquigarrow t'$ if an instance of t can be rewritten to t' by some rule $l \longrightarrow r$
  - defines symbolic transitions between logical terms
- Example

```
rl [wake]: N ; M ; [K,idle] PS => s N ; M ; [K,wait(N)] PS .
```

  - no rewriting from `N ; N ; PS`, but its instance `N ; N ; [K,idle] PS` can be rewritten

Computing Symbolic State Space

- Narrowing $\rightsquigarrow$
  - $t \rightsquigarrow t'$ if an instance of t can be rewritten to t' by some rule $l \longrightarrow r$
  - defines symbolic transitions between logical terms
- Example

```
rl [wake]: N ; M ; [K,idle] PS => s N ; M ; [K,wait(N)] PS .
```

  - no rewriting from `N ; N ; PS`, but its instance `N ; N ; [K,idle] PS` can be rewritten
  - `N ; N ; PS` $\rightsquigarrow$ `s N ; N ; [K,wait(N)] PS`

Computing Symbolic State Space

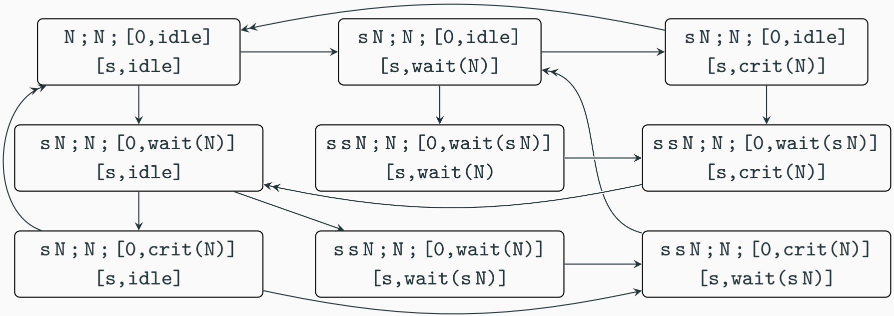
- Narrowing $\rightsquigarrow$
  - $t \rightsquigarrow t'$ if an instance of t can be rewritten to t' by some rule $l \longrightarrow r$
  - defines symbolic transitions between logical terms
- Example

```
rl [wake]: N ; M ; [K,idle] PS => s N ; M ; [K,wait(N)] PS .
```

  - no rewriting from `N ; N ; PS`, but its instance `N ; N ; [K,idle] PS` can be rewritten
  - `N ; N ; PS` $\rightsquigarrow$ `s N ; N ; [K,wait(N)] PS`
- Only need to compute the most general symbolic states

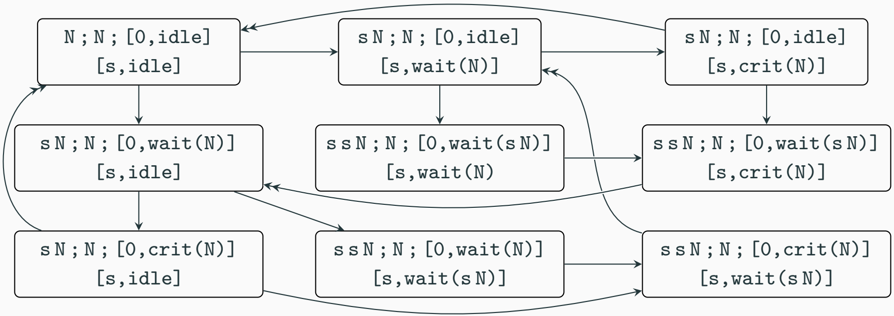
Example

- Bakery algorithm with two processes



Example

- Bakery algorithm with two processes



- Mutual exclusion can be verified for two processes!
- Can be verified for any number of processes, along with abstraction methods

Narrowing-Based Reachability Analysis Algorithm

$S \leftarrow \{t_{init}\}$;
$Queue$.enqueue(t_{init});
while $Queue \neq \emptyset$ **do**
    $u \leftarrow Queue$.dequeue();
    if *an instance of* u *matches* t_{goal} **then**
        return True
    foreach u' *such that* $u \rightsquigarrow u'$ **do**
        if u' *is a new symbolic state* **then**
            remove all instances of u' from S and $Queue$;
            $S \leftarrow S \cup \{u'\}$;
            $Queue$.enqueue(u')
return False

- Breadth-first search

Narrowing-Based Reachability Analysis Algorithm

$S \leftarrow \{t_{init}\}$;
$\mathcal{Queue}$.enqueue(t_{init});
while $\mathcal{Queue} \neq \emptyset$ **do**
  $u \leftarrow \mathcal{Queue}$.dequeue();
  if *an instance of u matches t_{goal}* **then**
    return True
  foreach u' *such that* $u \rightsquigarrow u'$ **do**
    if u' *is a new symbolic state* **then**
      remove all instances of u' from S and $\mathcal{Queue}$;
      $S \leftarrow S \cup \{u'\}$;
      $\mathcal{Queue}$.enqueue(u')
return False

- Breadth-first search
- Refutation-complete

Narrowing-Based Reachability Analysis Algorithm

```
S ← {t_init};
Queue.enqueue(t_init);
while Queue ≠ ∅ do
    u ← Queue.dequeue();
    if an instance of u matches t_goal then
        return True
    foreach u′ such that u ⇝ u′ do
        if u′ is a new symbolic state then
            remove all instances of u′ from
             S and Queue;
            S ← S ∪ {u′};
            Queue.enqueue(u′)
return False
```

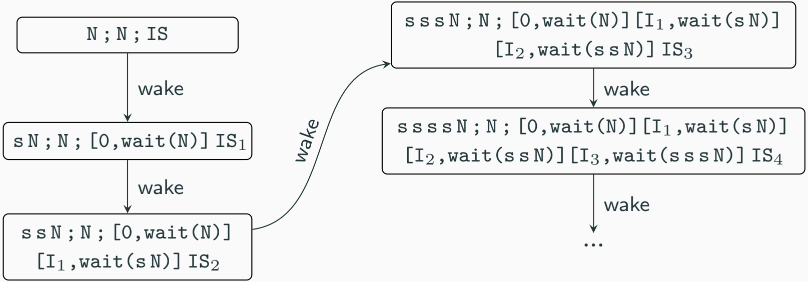
- Breadth-first search
- Refutation-complete
- Good at verifying the absence of errors (together with abstraction methods)

Limitation of the Existing Narrowing-Based Algorithm

- Bad at finding a counterexample

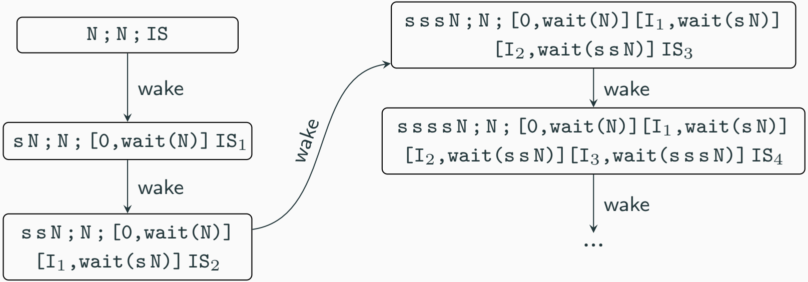
Limitation of the Existing Narrowing-Based Algorithm

- Bad at finding a counterexample
- BFS can generate a huge amount of useless symbolic states



Limitation of the Existing Narrowing-Based Algorithm

- Bad at finding a counterexample
- BFS can generate a huge amount of useless symbolic states



- # of symbolic states can increase dramatically as complexity of pattern increases

Heuristic Search For Narrowing-Based Reachability Analysis

```
S ← {t_init};
PriorityQ.enqueue(t_init);
while PriorityQ ≠ ∅ do
    u ← PriorityQ.dequeue();
    if an instance of u matches t_goal then
        return True
    foreach u′ such that u ⇝ u′ do
        if u′ is a new symbolic state then
            remove all instances of u′ from
              S and PriorityQ;
            S ← S ∪ {u′};
            PriorityQ.enqueue(u′)
return False
```

- Best-first search
- Priorities defined by scoring functions

Heuristic Search For Narrowing-Based Reachability Analysis

$S \leftarrow \{t_{init}\}$;
$PriorityQ.\mathtt{enqueue}(t_{init})$;
while $PriorityQ \neq \emptyset$ **do**
    $u \leftarrow PriorityQ.\mathtt{dequeue}()$;
    if *an instance of* u *matches* t_{goal} **then**
        return True
    foreach u' *such that* $u \rightsquigarrow u'$ **do**
        if u' *is a new symbolic state* **then**
            remove all instances of u' from S and $PriorityQ$;
            $S \leftarrow S \cup \{u'\}$;
            $PriorityQ.\mathtt{enqueue}(u')$
return False

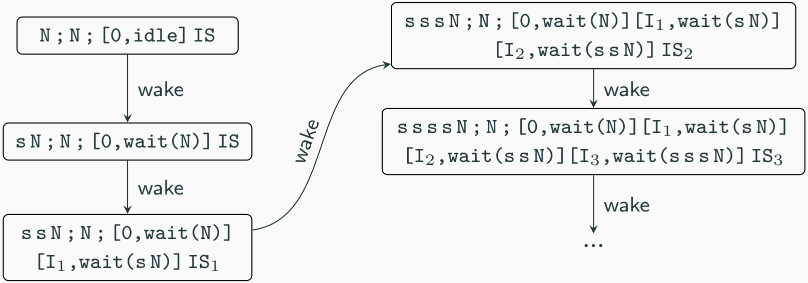
- Best-first search
- Priorities defined by scoring functions

Q: Effective scoring function for narrowing?

Q: Refutation-complete?

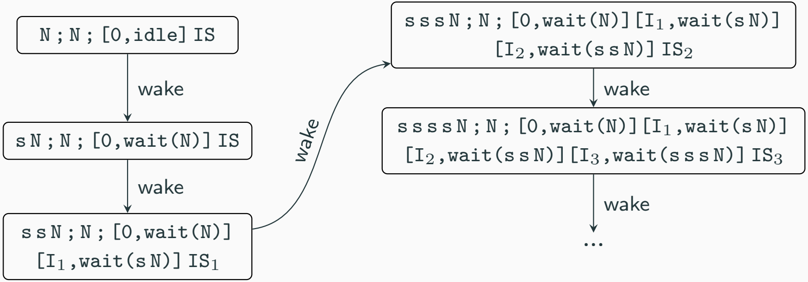
Observation

- # of symbolic states increases as complexity of pattern increases



Observation

- # of symbolic states increases as complexity of pattern increases



Goal

Minimize the state-space explosion due to the complexity of symbolic states.

Scoring Functions for Narrowing

- $score(t) = g(t) + h(t)$
  - a lower score indicates a higher priority

Scoring Functions for Narrowing

- $score(t) = g(t) + h(t)$
  - a lower score indicates a higher priority
- $g(t)$: the complexity of pattern t
  - e.g., the term size of t

Scoring Functions for Narrowing

- $score(t) = g(t) + h(t)$
  - a lower score indicates a higher priority
- $g(t)$: the complexity of pattern t
  - e.g., the term size of t
- $h(t)$: a heuristic estimation based on prior knowledge
  - e.g., the number of wait processes

A Caveat on Scoring Functions

- For the Bakery example, consider

$$score(t) = \text{the number of processes in } t$$

A Caveat on Scoring Functions

- For the Bakery example, consider

$$\color{blue}{score(t)} = \text{the number of processes in } t$$

- There exists an infinite narrowing sequence

$$\begin{tikzcd}
\texttt{0 ; 0 ; [0,idle] IS} \arrow[d, "wake"] & \texttt{s ; s ; [0,idle] IS} \arrow[d, "wake"] & \texttt{s s ; s s ; [0,idle] IS} \arrow[d, "wake"] \\
\texttt{s ; 0 ; [0,wait(0)] IS} \arrow[d, "crit"] & \texttt{s s ; s ; [0,wait(s)] IS} \arrow[d, "crit"] & \texttt{s s s ; s s ; [0,wait(s s)] IS} \arrow[d, "crit"] \\
\texttt{s ; 0 ; [0,crit(0) IS]} \arrow[uur, "exit"] & \texttt{s s ; s ; [0,crit(s)] IS} \arrow[uur, "exit"] & \cdots
\end{tikzcd}$$

A Caveat on Scoring Functions

- For the Bakery example, consider

$$\mathit{score}(t) = \text{the number of processes in } t$$

- There exists an infinite narrowing sequence

$$\begin{tikzcd}
\texttt{0 ; 0 ; [0,idle] IS} \arrow[d, "\mathit{wake}"] & \texttt{s ; s ; [0,idle] IS} \arrow[d, "\mathit{wake}"] & \texttt{s s ; s s ; [0,idle] IS} \arrow[d, "\mathit{wake}"] \\
\texttt{s ; 0 ; [0,wait(0)] IS} \arrow[d, "\mathit{crit}"] & \texttt{s s ; s ; [0,wait(s)] IS} \arrow[d, "\mathit{crit}"] & \texttt{s s s ; s s ; [0,wait(s s)] IS} \arrow[d, "\mathit{crit}"] \\
\texttt{s ; 0 ; [0,crit(0) IS]} \arrow[uur, "\mathit{exit}"] & \texttt{s s ; s ; [0,crit(s)] IS} \arrow[uur, "\mathit{exit}"] & \cdots
\end{tikzcd}$$

- Heuristic narrowing search will never visit symbolic states with higher scores!

Refutation-Completeness for Heuristic Narrowing Search

- There should be no infinite narrowing sequence with non-increasing scores

Refutation-Completeness for Heuristic Narrowing Search

- There should be no infinite narrowing sequence with non-increasing scores

Theorem

Let t_{init} be an initial pattern. For any score value c, if the set

$$\{u \mid t_{init} \rightsquigarrow^* u, score(u) \leq c\}$$

is finite, the heuristic narrowing search algorithm is refutation-complete.

Case Study: the OneThirdRule Consensus Algorithm

- OneThirdRule: a round-based distributed consensus algorithm

```
Initialization :
    x_p ← v_p;
Round r:
    broadcast x_p to all other nodes ;                    // init
    wait for messages from other nodes ;                  // waiting
    if |ReceivedMsg(r)| > 2/3 · N then
        x_p ← the smallest most frequently received value
        if more than 2/3 · N of the received values are x_p then
            Decide (x_p) ;                                // finished
        else
            proceed to Round r + 1;
```

- N node communicate via (possibly faulty) asynchronous message passing

Case Study: the OneThirdRule Consensus Algorithm

- OneThirdRule: a round-based distributed consensus algorithm

```
Initialization :
    x_p ← v_p;
Round r:
    broadcast x_p to all other nodes ;                    // init
    wait for messages from other nodes ;                  // waiting
    if |ReceivedMsg(r)| > 2/3 · N then
        x_p ← the smallest most frequently received value
        if more than 2/3 · N of the received values are x_p then
            Decide (x_p) ;                                // finished
        else
            proceed to Round r + 1;
```

- N node communicate via (possibly faulty) asynchronous message passing
- The algorithm uses $2/3$ as a decision threshold

OneThirdRule: Narrowing-Based Reachability Analysis

- Agreement requirement

  All nodes will agree on the same value when the algorithm terminates

OneThirdRule: Narrowing-Based Reachability Analysis

- Agreement requirement

  All nodes will agree on the same value when the algorithm terminates

- Try to find a counterexample of the agreement condition
  - with different decision threshold $1/2$ for any number of initial nodes

OneThirdRule: Narrowing-Based Reachability Analysis

- Agreement requirement

  All nodes will agree on the same value when the algorithm terminates

- Try to find a counterexample of the agreement condition
  - with different decision threshold $1/2$ for any number of initial nodes

- Failed to find a counterexample using the BFS-based algorithm
  - The number of symbolic states rapidly grows

Depth	1	2	3	4	5	6	$\cdots$
# States	4	26	88	318	1,248	5,030	$\cdots$

OneThirdRule: Heuristic Narrowing Search

- Two scoring functions

$$score_2(t) = numObjects(t)$$

$$score_3(t) = 100 \cdot numObjects(t) - numMessages(t) + 10 \cdot \sum_{obj \in objects(t)} \max(2, round(obj))$$

OneThirdRule: Heuristic Narrowing Search

- Two scoring functions

$$score_2(t) = numObjects(t)$$

$$score_3(t) = 100 \cdot numObjects(t) - numMessages(t) + 10 \cdot \sum_{obj \in objects(t)} \max(2, round(obj))$$

- Experimental results (T/O = 48 hours)

Search	Time (s)	# States
BFS	T/O	T/O
$score_2$	57,104	25,420
$score_3$	1,162	6,061

Ongoing and Future Work

- Rewriting logic으로 명세된 시스템의 패턴 기반 모델검증
  - symbolic state-space exploration using narrowing
  - good at verifying properties, but very bad at finding a counterexample
- 패턴 기반 모델검증을 위한 오류 탐색 전략
  - proved sufficient conditions to preserve refutation-completeness
  - demonstrated the power of our approach using OneThirdRule

Ongoing and Future Work

- Rewriting logic으로 명세된 시스템의 패턴 기반 모델검증
  - symbolic state-space exploration using narrowing
  - good at verifying properties, but very bad at finding a counterexample
- 패턴 기반 모델검증을 위한 오류 탐색 전략
  - proved sufficient conditions to preserve refutation-completeness
  - demonstrated the power of our approach using OneThirdRule
- 진행 연구
  - 패턴 기반 모델검증을 위한 효과적인 요약/추상화 기법
  - SMT를 활용한 패턴 기반 모델검증의 적용 범위 확대

Ongoing and Future Work

- Rewriting logic으로 명세된 시스템의 패턴 기반 모델검증
  - symbolic state-space exploration using narrowing
  - good at verifying properties, but very bad at finding a counterexample
- 패턴 기반 모델검증을 위한 오류 탐색 전략
  - proved sufficient conditions to preserve refutation-completeness
  - demonstrated the power of our approach using OneThirdRule
- 진행 연구
  - 패턴 기반 모델검증을 위한 효과적인 요약/추상화 기법
  - SMT를 활용한 패턴 기반 모델검증의 적용 범위 확대
- 향후 계획
  - 지진경보시스템 테스트 베드의 Promela 모델에 대한 패턴 기반 모델검증 적용
  - 패턴 기반 모델검증에 “재난오류 데이터베이스”를 활용하는 방안 연구

Thank you!