논리 기반 점진적 모델 검증

배경민

2022년 7월 14일

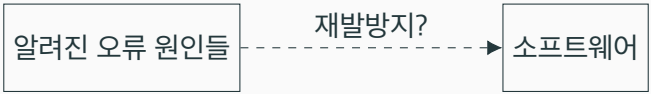
POSTECH 소프트웨어 검증 연구실

그룹3 연구목표: 소프트웨어재난 재발방지

소프트웨어

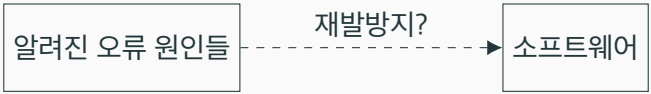
- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지

그룹3 연구목표: 소프트웨어재난 재발방지



- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축

그룹3 연구목표: 소프트웨어재난 재발방지



- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 의문1: 코드기반? 오류 분석이 완료된 동일한 특정 소프트웨어 구현 재검증?

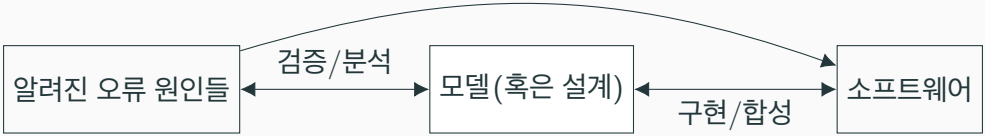
그룹3 연구목표: 소프트웨어재난 재발방지

- 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

그룹3 연구목표: 소프트웨어재난 재발방지

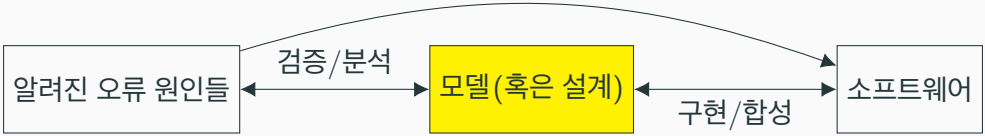
- 동일한 원인에 의해 발생하는 소프트웨어 재난의 재발을 방지
- 연구내용1: 알려진 오류 원인들에 대한 재난오류 데이터베이스 구축
- 연구내용2: 모델(혹은 설계) 단계에서 소프트웨어 재난의 원인을 분석
- 의문2: 모델이나 요구사항이 존재하지 않는 경우? 검증의 복잡성?

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

POSTECH 정형명세 및 모델검증 연구 (1)

- 접근방법: 논리 기반 모델 검증

Model		Logic System		Verification
시스템 명세		수학적 모델		
M	$\Longrightarrow$	$\mathcal{R}_M$	$\Longrightarrow$	모델검증
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)

- Signal Temporal Logic 모델 검증 연구 (포스터: 이지아 & 유근열)
  - 하이브리드 시스템 (Hybrid system): 소프트웨어 모델 + 물리모델
  - Signal Temporal Logic (STL): 연속적인 시그널의 성질을 표현

POSTECH 정형명세 및 모델검증 연구 (2)

- Signal Temporal Logic 모델 검증 연구 (포스터: 이지아 & 유근열)
  - 하이브리드 시스템 (Hybrid system): 소프트웨어 모델 + 물리모델
  - Signal Temporal Logic (STL): 연속적인 시그널의 성질을 표현
- 다양한 도메인에 대하여 논리 기반 모델 검증 적용
  - PLC (Programmable Logic Controller) ST 모델 (포스터: 이재서 & 김상기)
  - AADL 언어 기반 분산 시스템 모델 (포스터: 이재훈)
  - Trusted execution environment (TEE) API 모델 (포스터: 유근열 & 채승현)
  - OSEK/VDX OS API의 객체지향 정형명세 (김상기)

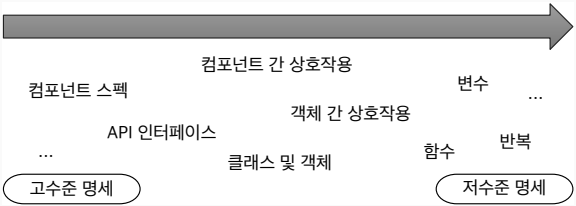
POSTECH 정형명세 및 모델검증 연구 (2)

- Signal Temporal Logic 모델 검증 연구 (포스터: 이지아 & 유근열)
  - 하이브리드 시스템 (Hybrid system): 소프트웨어 모델 + 물리모델
  - Signal Temporal Logic (STL): 연속적인 시그널의 성질을 표현
- 다양한 도메인에 대하여 논리 기반 모델 검증 적용
  - PLC (Programmable Logic Controller) ST 모델 (포스터: 이재서 & 김상기)
  - AADL 언어 기반 분산 시스템 모델 (포스터: 이재훈)
  - Trusted execution environment (TEE) API 모델 (포스터: 유근열 & 채승현)
  - OSEK/VDX OS API의 객체지향 정형명세 (김상기)
- Deep neural network (DNN) 검증
  - DNN 검증을 위한 요약 해석 기법 연구 (포스터: 연주은)
  - DNN 검증 시 발생하는 conflict 정보를 이용한 성능 향상 기법 연구 (포스터: 채승현)

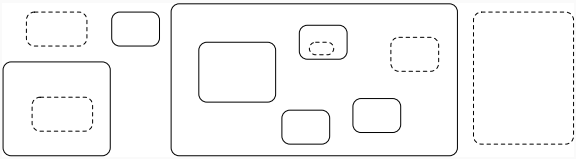
진행연구: 논리 기반 점진적 모델 검증

점진적 모델검증 연구의 필요성

- 고수준/저수준 정형명세

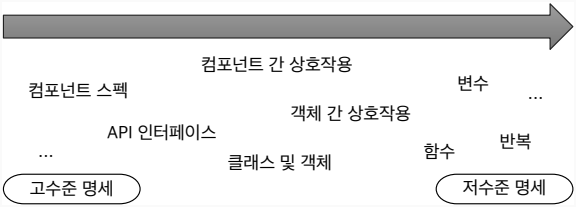


- 부분적으로 명세되는 정형모델

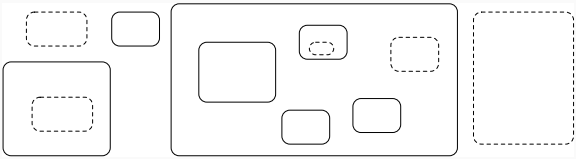


점진적 모델검증 연구의 필요성

- 고수준/저수준 정형명세



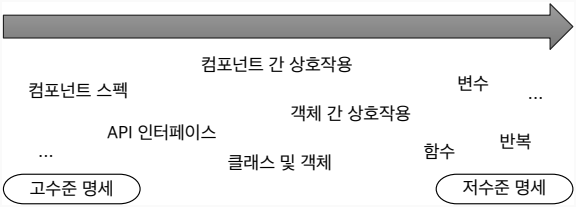
- 부분적으로 명세되는 정형모델



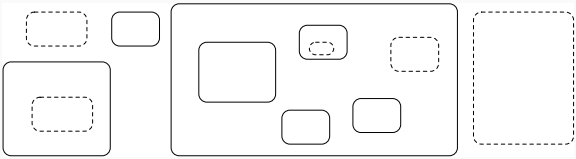
- Q1: 명세 수준에 따른 요약?
  - cf. 데이터 요약

점진적 모델검증 연구의 필요성

- 고수준/저수준 정형명세



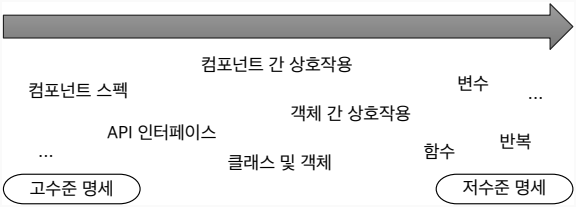
- 부분적으로 명세되는 정형모델



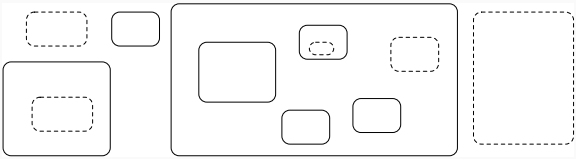
- Q1: 명세 수준에 따른 요약?
  - cf. 데이터 요약
- Q2: 부분적 정형모델의 검증?
  - interface의 spec 사용

점진적 모델검증 연구의 필요성

- 고수준/저수준 정형명세



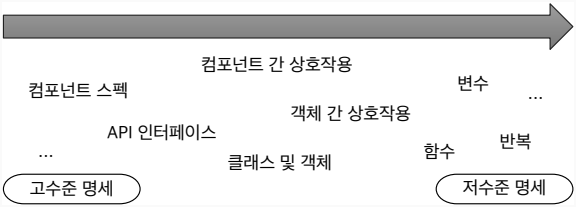
- 부분적으로 명세되는 정형모델



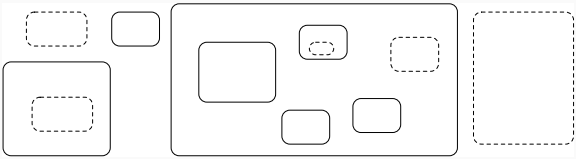
- Q1: 명세 수준에 따른 요약?
  - cf. 데이터 요약
- Q2: 부분적 정형모델의 검증?
  - interface의 spec 사용
- Q3: 이전 정형검증 결과 활용?
  - 비슷한 오류 방지에 유용함

점진적 모델검증 연구의 필요성

- 고수준/저수준 정형명세



- 부분적으로 명세되는 정형모델

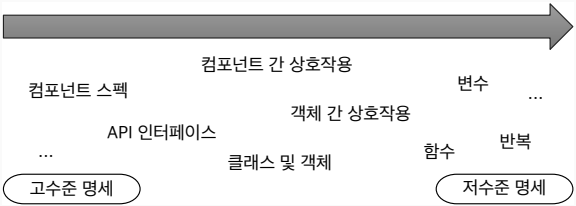


- Q1: 명세 수준에 따른 요약?
  - cf. 데이터 요약
- Q2: 부분적 정형모델의 검증?
  - interface의 spec 사용
- Q3: 이전 정형검증 결과 활용?
  - 비슷한 오류 방지에 유용함

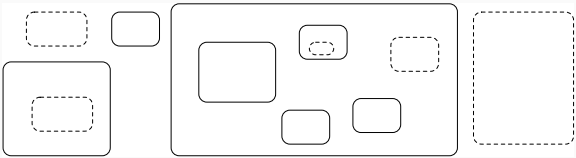
⇒ 정리증명: 손쉽게(?) 지원
  - lemma, assumption, ⋯

점진적 모델검증 연구의 필요성

- 고수준/저수준 정형명세



- 부분적으로 명세되는 정형모델



- Q1: 명세 수준에 따른 요약?
  - cf. 데이터 요약
- Q2: 부분적 정형모델의 검증?
  - interface의 spec 사용
- Q3: 이전 정형검증 결과 활용?
  - 비슷한 오류 방지에 유용함

⇒ 정리증명: 손쉽게(?) 지원
- lemma, assumption, ⋯

⇒ 모델검증: 잘 지원되지 않음
- 부분적 검증 결과 활용 X

점진적 모델검증의 어려움

- 모델검증: 시스템의 모든 가능한 상태를 확인
  - 자동화, 오류 재현 용이, 오류없음 증명 가능, … / 상태폭발문제, …

점진적 모델검증의 어려움

- 모델검증: 시스템의 모든 가능한 상태를 확인
  - 자동화, 오류 재현 용이, 오류없음 증명 가능, … / 상태폭발문제, …
- 전제조건: (정형)모델이 "실행가능" 해야 함
  - 부분적 혹은 고수준 정형모델은 보통 실행가능하지 않음

점진적 모델검증의 어려움

- 모델검증: 시스템의 모든 가능한 상태를 확인
  - 자동화, 오류 재현 용이, 오류없음 증명 가능, ⋯ / 상태폭발문제, ⋯
- 전제조건: (정형)모델이 "실행가능" 해야 함
  - 부분적 혹은 고수준 정형모델은 보통 실행가능하지 않음
- Dummy/Stub 컴포넌트 활용??

점진적 모델검증의 어려움

- 모델검증: 시스템의 모든 가능한 상태를 확인
  - 자동화, 오류 재현 용이, 오류없음 증명 가능, ⋯ / 상태폭발문제, ⋯
- 전제조건: (정형)모델이 "실행가능" 해야 함
  - 부분적 혹은 고수준 정형모델은 보통 실행가능하지 않음
- Dummy/Stub 컴포넌트 활용??
  - 가능한 행위의 일부만 해당하여 모델검증의 soundness가 유지되지 않음

연구목표: 점진적 모델검증

- 수준 별 및 부분적 정형명세
  - 고수준 인터페이스에서부터 명세하여, 점진적으로 구체적인 상태나 행위를 추가 가능

연구목표: 점진적 모델검증

- 수준 별 및 부분적 정형명세
  - 고수준 인터페이스에서부터 명세하여, 점진적으로 구체적인 상태나 행위를 추가 가능
  - 보통 많은 정형명세 언어에서 이와 같은 명세를 허용함

연구목표: 점진적 모델검증

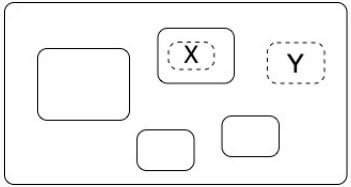
- 수준 별 및 부분적 정형명세
  - 고수준 인터페이스에서부터 명세하여, 점진적으로 구체적인 상태나 행위를 추가 가능
  - 보통 많은 정형명세 언어에서 이와 같은 명세를 허용함
- 고수준/부분적 모델에 대한 sound한 모델검증 수행
  - 고수준/부분적 모델에서 요구사항 성립 $\Rightarrow$ 관련된 모든 "구체화" 된 모델에서 요구사항 성립

연구목표: 점진적 모델검증

- 수준 별 및 부분적 정형명세
  - 고수준 인터페이스에서부터 명세하여, 점진적으로 구체적인 상태나 행위를 추가 가능
  - 보통 많은 정형명세 언어에서 이와 같은 명세를 허용함
- 고수준/부분적 모델에 대한 sound한 모델검증 수행
  - 고수준/부분적 모델에서 요구사항 성립 ⇒ 관련된 모든 "구체화" 된 모델에서 요구사항 성립
  - 모델검증 도구에서 보통 지원되지 않음

Main Idea

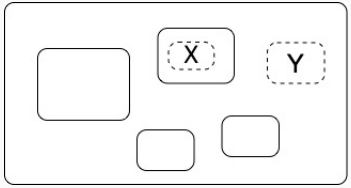
- 시스템 정형명세
  - 시스템 상태: 아직 구체화되지 않은 부분을 "논리적인 변수들" 및 이에 대한 제약조건으로 표현



$\| \ \phi_1(X) \wedge \phi_2(Y)$

Main Idea

- 시스템 정형명세
  - 시스템 상태: 아직 구체화되지 않은 부분을 "논리적인 변수들" 및 이에 대한 제약조건으로 표현

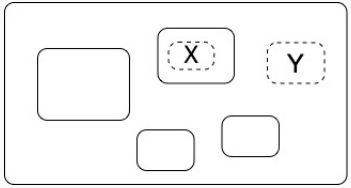


$\| \ \phi_1(X) \wedge \phi_2(Y)$

  - 행위: 아직 구체화되지 않은 부분은 제약조건을 만족하는 "임의의 모든 행위"를 가진다고 정의

Main Idea

- 시스템 정형명세
  - 시스템 상태: 아직 구체화되지 않은 부분을 "논리적인 변수들" 및 이에 대한 제약조건으로 표현



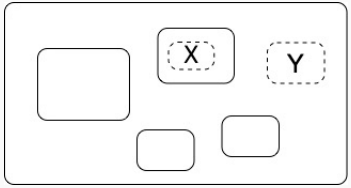
$\| \ \phi_1(X) \wedge \phi_2(Y)$

  - 행위: 아직 구체화되지 않은 부분은 제약조건을 만족하는 "임의의 모든 행위"를 가진다고 정의

$\Rightarrow$ 정리증명 기반 모델링 및 일부 정형분석 도구(Alloy)에서 사용되는 접근방법

Main Idea

- 시스템 정형명세
  - 시스템 상태: 아직 구체화되지 않은 부분을 "논리적인 변수들" 및 이에 대한 제약조건으로 표현



$\| \ \phi_1(X) \wedge \phi_2(Y)$

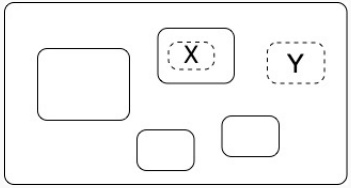
  - 행위: 아직 구체화되지 않은 부분은 제약조건을 만족하는 "임의의 모든 행위"를 가진다고 정의

$\Rightarrow$ 정리증명 기반 모델링 및 일부 정형분석 도구(Alloy)에서 사용되는 접근방법

- 모델검증 알고리즘
  - 변수와 제약조건을 포함하는 상태에 대한 논리 기반 "symbolic execution" 수행

Main Idea

- 시스템 정형명세
  - 시스템 상태: 아직 구체화되지 않은 부분을 "논리적인 변수들" 및 이에 대한 제약조건으로 표현



$\| \ \phi_1(X) \wedge \phi_2(Y)$

  - 행위: 아직 구체화되지 않은 부분은 제약조건을 만족하는 "임의의 모든 행위"를 가진다고 정의

$\Rightarrow$ 정리증명 기반 모델링 및 일부 정형분석 도구(Alloy)에서 사용되는 접근방법

- 모델검증 알고리즘
  - 변수와 제약조건을 포함하는 상태에 대한 논리 기반 "symbolic execution" 수행
  - Decidability를 위한 모델의 크기 기준 bounded model checking

진행연구: Rewriting 기반 점진적 정형명세 및 모델검증 연구

- Rewriting 기반 정형명세 프레임워크 연구

진행연구: Rewriting 기반 점진적 정형명세 및 모델검증 연구

- Rewriting 기반 정형명세 프레임워크 연구
  - 구체화되지 않은 부분이 "임의의 모든 행위"를 가지도록 어떻게 정의할 것인가?

진행연구: Rewriting 기반 점진적 정형명세 및 모델검증 연구

- Rewriting 기반 정형명세 프레임워크 연구
  - 구체화되지 않은 부분이 "임의의 모든 행위"를 가지도록 어떻게 정의할 것인가?
  - 고수준/부분적 모델과 구체화된 모델 간의 관계는?

진행연구: Rewriting 기반 점진적 정형명세 및 모델검증 연구

- Rewriting 기반 정형명세 프레임워크 연구
  - 구체화되지 않은 부분이 "임의의 모든 행위"를 가지도록 어떻게 정의할 것인가?
  - 고수준/부분적 모델과 구체화된 모델 간의 관계는?
- Narrowing 기반 symbolic execution 모델검증 알고리즘 (강병지)
  - narrowing: term rewriting에 대한 symbolic execution 기술 중 하나
  - term의 크기 기준 bounded narrowing 기술 연구 및 구현

배경지식: Rewriting Logic 기반 정형 명세

- Rewriting logic: 높은 표현력을 가진 명세 언어로 다양한 모델링 언어의 의미 정의 가능

  동시성 모델: actors, process calculi, Petri nets, ⋯

  프로그래밍 언어: C, Java, JavaScript, Scheme, Python, ⋯

  디자인 언어: Verilog, ABEL, AADL, Ptolemy II, Orc, ⋯

배경지식: Rewriting Logic 기반 정형 명세

- Rewriting logic: 높은 표현력을 가진 명세 언어로 다양한 모델링 언어의 의미 정의 가능

  동시성 모델: actors, process calculi, Petri nets, ⋯

  프로그래밍 언어: C, Java, JavaScript, Scheme, Python, ⋯

  디자인 언어: Verilog, ABEL, AADL, Ptolemy II, Orc, ⋯

- 시스템 상태: 대수적 자료 구조
  - recursive data types and functions
  - lists, sets, multi-sets, ⋯

배경지식: Rewriting Logic 기반 정형 명세

- Rewriting logic: 높은 표현력을 가진 명세 언어로 다양한 모델링 언어의 의미 정의 가능

  동시성 모델: actors, process calculi, Petri nets, ⋯

  프로그래밍 언어: C, Java, JavaScript, Scheme, Python, ⋯

  디자인 언어: Verilog, ABEL, AADL, Ptolemy II, Orc, ⋯

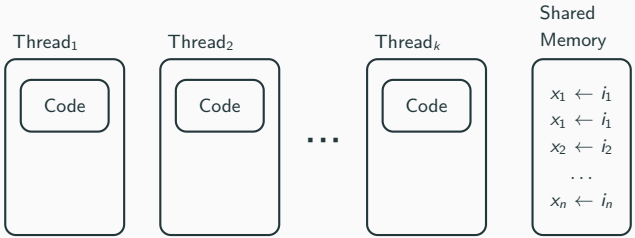
- 시스템 상태: 대수적 자료 구조
  - recursive data types and functions
  - lists, sets, multi-sets, ⋯
- 시스템의 상태 변화
  - rewrite rule $t \rightarrow t'$
  - 패턴 t에서 패턴 t'으로의 변화

예: 간단한 프로그래밍 언어

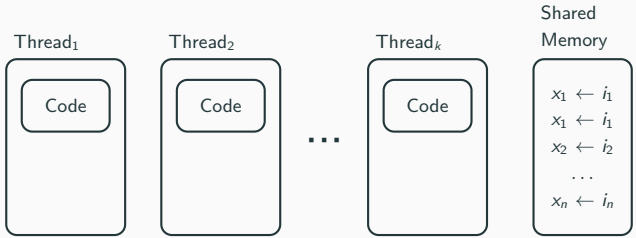
```
vars n, i, s;
n := 100;
i := 0;
s := 0;
while i <= n do
    s := s + i;
    i := i + 1
```

$$
\begin{aligned}
\mathit{Expression} ::= {} & \mathit{Integer} \\
| \; & \mathit{Expression} + \mathit{Expression} \\
| \; & \mathit{Expression} * \mathit{Expression} \\
| \; & \mathit{Expression} == \mathit{Expression} \\
| \; & \mathit{Expression} \; != \mathit{Expression} \\
| \; & (\mathit{Expression}) \\
\mathit{Program} ::= {} & \mathsf{skip} \\
| \; & \mathit{Program} \; ; \; \mathit{Program} \\
| \; & \mathit{Variable} = \mathit{Expression} \\
| \; & \mathsf{if} \; (\mathit{Expression}) \; \{ \; \mathit{Program} \; \} \\
| \; & \mathsf{while} \; (\mathit{Expression}) \; \{ \; \mathit{Program} \; \}
\end{aligned}
$$

예: 간단한 프로그래밍 언어의 실행 상태



예: 간단한 프로그래밍 언어의 실행 상태



- 대수적 자료구조로의 표현:

$$\{[1, Code_1] \mid [2, Code_2] \mid \cdots \mid [k, Code_k], \quad [x_1, i_1]\,[x_2, i_2] \cdots [x_n, i, n]\}$$

시스템 상태의 Pattern Matching

- 시스템 상태

```
{ [1, c1 = 1 ; while (c2 == 1) { ... } ; ...]   |   [2, if (turn == 1) { ... } ; ...],
   [c1,0] [c2,0] [turn,1] }
```

시스템 상태의 Pattern Matching

- 시스템 상태

```
{ [1, c1 = 1 ; while (c2 == 1) { ... } ; ...]   |   [2, if (turn == 1) { ... } ; ...],
   [c1,0] [c2,0] [turn,1] }
```

- 패턴

$$\{[\, I\,, V\ =\ E\ ;\ P\,] \mid \mathit{THREADS}\,, \mathit{MEM}\,\}$$

시스템 상태의 Pattern Matching

- 시스템 상태

```
{ [1, c1 = 1 ; while (c2 == 1) { ... } ; ...]   |   [2, if (turn == 1) { ... } ; ...],
  [c1,0] [c2,0] [turn,1] }
```

- 패턴

$$\{[\, I\,, V\ =\ E\ ;\ P\,] \mid \mathit{THREADS}\,, \mathit{MEM}\,\}$$

- 패턴 매칭

$$I \mapsto 1 \qquad V \mapsto c1 \qquad E \mapsto 1 \qquad P \mapsto \text{while}\ (\text{c2} == 1)\ \ldots\ ;\ \ldots$$

$$\mathit{THREADS} \mapsto [2,\ \text{if}\ (\text{turn} == 1)\ \text{c2} = 0\ ;\ \ldots\ ;\ \ldots]$$

$$\mathit{MEM} \mapsto [\text{c1}\,,0]\ \ [\text{c2}\,,0]\ \ [\text{turn}\,,1]$$

예: 간단한 프로그래밍 언어의 Operational Semantics

$$\{[I, \mathit{skip}\ ;\ P] \mid \mathit{THREADS}, \mathit{MEM}\}$$
$$\rightarrow\ \{[I, P] \mid \mathit{THREADS}, \mathit{MEM}\}$$

예: 간단한 프로그래밍 언어의 Operational Semantics

$$\{[I, skip\ ;\ P] \mid THREADS, MEM\}$$
$$\rightarrow\ \{[I, P] \mid THREADS, MEM\}$$

$$\{[I, (V = E)\ ;\ P] \mid THREADS, MEM\}$$
$$\rightarrow\ \{[I, (V = E)\ ;\ P] \mid THREADS, update(MEM, [V, eval(E)])\}$$

예: 간단한 프로그래밍 언어의 Operational Semantics

$$
\begin{aligned}
& \{[I, \mathit{skip}\ ;\ P] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\rightarrow\ & \{[I, P] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, (V = E)\ ;\ P] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\rightarrow\ & \{[I, (V = E)\ ;\ P] \mid \mathit{THREADS}, \mathit{update}(\mathit{MEM}, [V, \mathit{eval}(E)])\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, \text{if}\ (T)\ \{P\}\ ;\ P'] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\rightarrow\ & \{[I, \big(\mathit{eval}(T)\ ?\ P\ :\ \mathit{skip}\big)\ ;\ P'] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

예: 간단한 프로그래밍 언어의 Operational Semantics

$$
\begin{aligned}
& \{[I, \mathit{skip}\ ;\ P] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\rightarrow\ & \{[I, P] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, (V = E)\ ;\ P] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\rightarrow\ & \{[I, (V = E)\ ;\ P] \mid \mathit{THREADS}, \mathit{update}(\mathit{MEM}, [V, \mathit{eval}(E)])\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, \text{if}\ (T)\ \{P\}\ ;\ P'] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\rightarrow\ & \{[I, \big(\mathit{eval}(T)\ ?\ P\ :\ \mathit{skip}\big)\ ;\ P'] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, \text{while}\ (T)\ \{P\}\ ;\ P'] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\rightarrow\ & \{[I, \big(\mathit{eval}(T)\ ?\ (P\ ;\ \text{while}\ (T)\ \{P\})\ :\ \mathit{skip}\big)\ ;\ P'] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

Rewriting 기반 고수준/부분적 정형 명세

- 시스템 상태: algebraic data structure with "holes"
  - 구체화되지 않은 부분은 논리적 변수로 표현

Rewriting 기반 고수준/부분적 정형 명세

- 시스템 상태: algebraic data structure with "holes"
  - 구체화되지 않은 부분은 논리적 변수로 표현
  - Logical term $t(x_1, \ldots, x_n)$: 가능한 모든 "pattern" t의 instance 표현

Rewriting 기반 고수준/부분적 정형 명세

- 시스템 상태: algebraic data structure with "holes"
  - 구체화되지 않은 부분은 논리적 변수로 표현
  - Logical term $t(x_1, \ldots, x_n)$: 가능한 모든 "pattern" t의 instance 표현
- 시스템의 상태 변화: rewrite rule $t \to t'$
  - 우측 t'은 좌측 t에 나타나지 않은 변수를 활용하여 임의의 행위 표현

Rewriting 기반 고수준/부분적 정형 명세

- 시스템 상태: algebraic data structure with "holes"
  - 구체화되지 않은 부분은 논리적 변수로 표현
  - Logical term $t(x_1, \ldots, x_n)$: 가능한 모든 "pattern" t의 instance 표현
- 시스템의 상태 변화: rewrite rule $t \to t'$
  - 우측 t'은 좌측 t에 나타나지 않은 변수를 활용하여 임의의 행위 표현
  - $t(x_1, \ldots, x_n) \to t'(x_1, \ldots, x_n, y_1, \ldots, y_n)$

예: 간단한 프로그래밍 언어

- 시스템 상태

```
{ [1, UNKNOWN ; while (c2 == 1) { ... } ; ...] | [2, if (turn == 1) { ... } ; ...] | PROCS,
  [c1,0] [c2,0] [turn,1] MEM }
```

  - 논리적 변수를 이용하여 구체화되지 않은 부분 표현

예: 간단한 프로그래밍 언어

- 시스템 상태

```
{ [1, UNKNOWN ; while (c2 == 1) { ... } ; ...] | [2, if (turn == 1) { ... } ; ...] | PROCS,
  [c1,0] [c2,0] [turn,1] MEM }
```

  - 논리적 변수를 이용하여 구체화되지 않은 부분 표현

- 시스템의 상태 변화

$$\{[I, \mathit{UNKNOWN} ; P] \mid \mathit{THREADS}, [X, \mathit{VAL}]\ \mathit{MEM}\}$$
$$\rightarrow \{[I, P] \mid \mathit{THREADS}, [X, \mathit{VAL'}]\ \mathit{MEM}\}$$

  - 우측에 새로운 논리적 변수를 이용하여 임의의 행위 표현

Narrowing 및 Bounded Narrowing

- Narrowing: 변수가 있는 logical term 간의 symbolic execution 정의
  - 위에서 제안한 정형모델에 대한 symbolic model checking 가능

Narrowing 및 Bounded Narrowing

- Narrowing: 변수가 있는 logical term 간의 symbolic execution 정의
  - 위에서 제안한 정형모델에 대한 symbolic model checking 가능
- Narrowing은 일반적으로 undecidable
  - 예: 특정 pattern에서 무한하게 많은 "상이한" 행위가 가능한 경우

Narrowing 및 Bounded Narrowing

- Narrowing: 변수가 있는 logical term 간의 symbolic execution 정의
  - 위에서 제안한 정형모델에 대한 symbolic model checking 가능
- Narrowing은 일반적으로 undecidable
  - 예: 특정 pattern에서 무한하게 많은 "상이한" 행위가 가능한 경우
- 제안기술: bounded narrowing
  - 주어진 term/pattern size bound 대비 완전한 알고리즘 연구
  - Maude 도구에 prototype 알고리즘 구현

Ongoing and Future Work

- 현재 프로토타입 구현은 오직 제약조건이 없는 pattern에 대한 경우만 고려
  - SMT 조건 등으로 제약되는 pattern에 대한 bounded constrained narrowing 기술 연구

Ongoing and Future Work

- 현재 프로토타입 구현은 오직 제약조건이 없는 pattern에 대한 경우만 고려
  - SMT 조건 등으로 제약되는 pattern에 대한 bounded constrained narrowing 기술 연구
- 제안된 기술을 활용한 점진적 모델검증 사례연구
  - OS API 모델 등에 적용

Ongoing and Future Work

- 현재 프로토타입 구현은 오직 제약조건이 없는 pattern에 대한 경우만 고려
  - SMT 조건 등으로 제약되는 pattern에 대한 bounded constrained narrowing 기술 연구
- 제안된 기술을 활용한 점진적 모델검증 사례연구
  - OS API 모델 등에 적용
- 점진적인 정형명세 과정에서 상위 수준의 모델검증 결과를 재활용하는 방법 연구
  - 비슷한 오류를 찾기 위해 이전 모델검증 결과 활용 가능

Ongoing and Future Work

- 현재 프로토타입 구현은 오직 제약조건이 없는 pattern에 대한 경우만 고려
  - SMT 조건 등으로 제약되는 pattern에 대한 bounded constrained narrowing 기술 연구
- 제안된 기술을 활용한 점진적 모델검증 사례연구
  - OS API 모델 등에 적용
- 점진적인 정형명세 과정에서 상위 수준의 모델검증 결과를 재활용하는 방법 연구
  - 비슷한 오류를 찾기 위해 이전 모델검증 결과 활용 가능
- 구체화된 모델을 위와 반대로 요약하여 모델검증 성능 향상 기법 연구
  - 오류 데이터베이스/패턴 기반

Thank you!