

프로그래밍 언어의 의미

문순원

December 21, 2022

소프트웨어 안정성을 높이는 방법

테스팅

- ▶ 몇 가지 케이스에서 프로그램의 실행 결과를 기대되는 결과와 비교
- ▶ 테스트를 통과했더라도 버그가 없음을 확신할 수는 없음

소프트웨어 안정성을 높이는 방법

테스팅

- ▶ 몇 가지 케이스에서 프로그램의 실행 결과를 기대되는 결과와 비교
- ▶ 테스트를 통과했더라도 버그가 없음을 확신할 수는 없음

프로그램 분석

- ▶ 프로그램을 자동으로 분석하는 프로그램
- ▶ 정지문제에 따른 한계

소프트웨어 안정성을 높이는 방법

테스팅

- ▶ 몇 가지 케이스에서 프로그램의 실행 결과를 기대되는 결과와 비교
- ▶ 테스트를 통과했더라도 버그가 없음을 확신할 수는 없음

프로그램 분석

- ▶ 프로그램을 자동으로 분석하는 프로그램
- ▶ 정지문제에 따른 한계

모델 체킹

- ▶ 상태 전이 기계의 특정 성질을 자동으로 검증
- ▶ State explosion problem을 피하기 쉽지 않음

소프트웨어 안정성을 높이는 방법

테스팅

- ▶ 몇 가지 케이스에서 프로그램의 실행 결과를 기대되는 결과와 비교
- ▶ 테스트를 통과했더라도 버그가 없음을 확신할 수는 없음

프로그램 분석

- ▶ 프로그램을 자동으로 분석하는 프로그램
- ▶ 정지문제에 따른 한계

모델 체킹

- ▶ 상태 전이 기계의 특정 성질을 자동으로 검증
- ▶ State explosion problem을 피하기 쉽지 않음

정형 검증

- ▶ 사람이 직접 프로그램의 성질을 증명
- ▶ 굉장히 높은 비용

정형 검증

정형 검증 (Formal verification)

the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics
— from Wikipedia

정형 검증

정형 검증 (Formal verification)

the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics
— from Wikipedia

- ▶ 프로그램과 그 스펙을 수학적 대상으로 다룸
- ▶ 연역적 방법을 사용해 프로그램의 성질을 증명
- ▶ 증명의 무결성을 컴퓨터로 검사할 수 있음

프로그램 정의의 계층

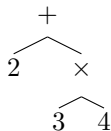
- ▶ 기호들의 나열 (concrete syntax)

[‘2’, ‘+’, ‘3’, ‘×’, ‘4’]

프로그램 정의의 계층

- ▶ 기호들의 나열 (concrete syntax)
- ▶ 추상 구문 트리 (abstract syntax)

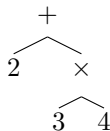
[‘2’, ‘+’, ‘3’, ‘×’, ‘4’]



프로그램 정의의 계층

- ▶ 기호들의 나열 (concrete syntax)
- ▶ 추상 구문 트리 (abstract syntax)

[‘2’, ‘+’, ‘3’, ‘×’, ‘4’]

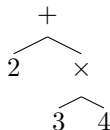


- ▶ 의미 (semantics)
프로그래밍 언어의 패러다임, 의미의 용도에 따라 여러가지 접근이 있다.

프로그램 정의의 계층

- ▶ 기호들의 나열 (concrete syntax)
- ▶ 추상 구문 트리 (abstract syntax)

[‘2’, ‘+’, ‘3’, ‘×’, ‘4’]



- ▶ 의미 (semantics)
프로그래밍 언어의 패러다임, 의미의 용도에 따라 여러가지 접근이 있다.
 - ▶ Operational semantics
 - ▶ Big-step semantics
 - ▶ Small-step semantics
 - ▶ Denotational semantics
 - ▶ Axiomatic semantics

Imp

간단한 프로그래밍 언어 Imp를 통해 추상 문법과 의미를 정의하는 방법을 알아보자.

Imp

간단한 프로그래밍 언어 Imp를 통해 추상 문법과 의미를 정의하는 방법을 알아보자.

Id	x, y, z	
AExp	E	$::= n \mid x \mid E_1 + E_2 \mid -E \mid E_1 \times E_2 \mid \dots$
BExp	B	$::= \text{true} \mid \text{false} \mid \neg B \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid$ $E_1 = E_2 \mid E_1 \leq E_2 \mid \dots$
Stmt	S	$::= \text{skip} \mid x \leftarrow E \mid S_1; S_2 \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \mid \text{while } B$

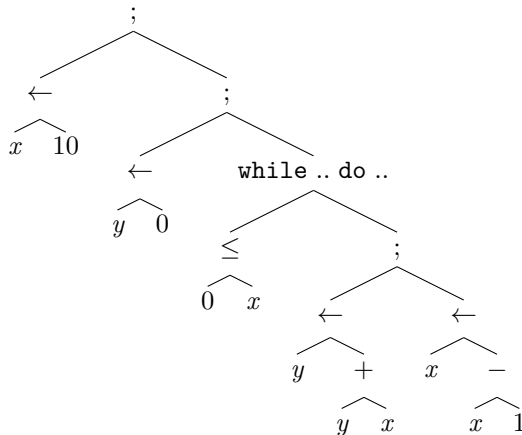
- ▶ 정수와 논리 연산이 가능한 명령형 언어
- ▶ 변수는 정수만을 담을 수 있다
- ▶ 제어 구문은 if, while 뿐
- ▶ AExp, BExp, Stmt는 귀납적으로 정의된 추상 문법 트리들의 집합

Imp

10이하 자연수들의 합을 구하는 프로그램

$$x \leftarrow 10; y \leftarrow 0; \text{while } 0 \leq x \text{ do } \{y \leftarrow y + x; x \leftarrow x - 1\}$$

10이하 자연수들의 합을 구하는 프로그램

$$x \leftarrow 10; y \leftarrow 0; \text{while } 0 \leq x \text{ do } \{y \leftarrow y + x; x \leftarrow x - 1\}$$


메모리 상태

명령형 프로그래밍 언어는 변수의 상태를 바꿈으로서 계산을 수행하기 때문에, 이러한 언어의 의미를 다루기 위해서는 먼저 메모리를 정의할 필요가 있다. Imp의 메모리 상태는 다음처럼 정의할 수 있다.

$$M \in \text{Mem} = \mathbb{Z}^{\text{Id}}$$

메모리 상태

메모리의 상태가 정해져있다면 AExp와 BExp의 값을 구할 수 있다.

$$\text{eval} : \text{Mem} \times \text{AExp} \rightarrow \mathbb{Z}$$

$$\text{eval} : \text{Mem} \times \text{BExp} \rightarrow \mathbb{B}$$

$$\text{eval}(M, n) = n$$

$$\text{eval}(M, x) = M(x)$$

$$\text{eval}(M, E_1 + E_2) = \text{eval}(M, E_1) + \text{eval}(M, E_2)$$

...

$$\text{eval}(M, \text{true}) = \text{T}$$

$$\text{eval}(M, \text{false}) = \text{F}$$

$$\text{eval}(M, B_1 \wedge B_2) = \text{eval}(M, B_1) \wedge \text{eval}(M, B_2)$$

$$\text{eval}(M, E_1 = E_2) = \begin{cases} \text{T}, & \text{if } \text{eval}(M, E_1) = \text{eval}(M, E_2) \\ \text{F}, & \text{otherwise} \end{cases}$$

...

Imp의 의미 - Big-step semantics

명령형 언어의 의미를 메모리의 상태를 바꾸는 함수로 이해할 수 있지 않을까?

$$\text{Stmt} \times \text{Mem} \rightarrow \text{Mem}$$

Imp의 의미 - Big-step semantics

명령형 언어의 의미를 메모리의 상태를 바꾸는 함수로 이해할 수 있지 않을까?

$$\text{Stmt} \times \text{Mem} \rightarrow \text{Mem}$$

- ▶ 프로그램이 종료하지 않는다면?
- ▶ 프로그램이 비결정적이라면?

Imp의 의미 - Big-step semantics

명령형 언어의 의미를 메모리의 상태를 바꾸는 함수로 이해할 수 있지 않을까?

$$\text{Stmt} \times \text{Mem} \rightarrow \text{Mem}$$

- ▶ 프로그램이 종료하지 않는다면?
- ▶ 프로그램이 비결정적이라면?

명령형 언어의 의미를 메모리의 상태를 바꾸는 ‘관계’로 정의하자.

$$\text{Stmt} \rightarrow \mathcal{P}(\text{Mem} \times \text{Mem})$$

$$M_1 \vdash S \Downarrow M_2$$

“상태 M_1 에서 프로그램 S 가 실행되었을 때 상태 M_2 로 종료할 수 있다”

Imp의 의미 - Big-step semantics

$$\overline{M \vdash \text{skip} \Downarrow M}$$

$$\frac{\text{eval}(M, E) = v}{M \vdash x \leftarrow E \Downarrow M[x \mapsto v]}$$

$$\frac{M_1 \vdash S_1 \Downarrow M_2 \quad M_2 \vdash S_2 \Downarrow M_3}{M_1 \vdash S_1; S_2 \Downarrow M_3}$$

$$\frac{\text{eval}(M_1, B) = \text{T} \quad M_1 \vdash S_1 \Downarrow M_2}{M_1 \vdash \text{if } B \text{ then } S_1 \text{ else } S_2 \Downarrow M_2}$$

$$\frac{\text{eval}(M, B) = \text{F} \quad M_1 \vdash S_2 \Downarrow M_2}{M_1 \vdash \text{if } B \text{ then } S_1 \text{ else } S_2 \Downarrow M_2}$$

$$\frac{\text{eval}(M, B) = \text{T} \quad M_1 \vdash S \Downarrow M_2 \quad M_2 \vdash \text{while } B \text{ do } S \Downarrow M_3}{M_1 \vdash \text{while } B \text{ do } S \Downarrow M_3}$$

$$\frac{\text{eval}(M, B) = \text{F}}{M \vdash \text{while } B \text{ do } S \Downarrow M}$$

Imp의 의미 - Big-step semantics

상태 M_1 에서 프로그램 S 가 무한루프를 도는 경우에는

$$\{M_2 \mid M_1 \vdash S \Downarrow M_2\} = \emptyset$$

Imp의 의미 - Big-step semantics

상태 M_1 에서 프로그램 S 가 무한루프를 도는 경우에는

$$\{M_2 \mid M_1 \vdash S \Downarrow M_2\} = \emptyset$$

- ▶ 무한루프를 도는 서로 다른 프로그램들을 구분할 수 없음
- ▶ 비결정적으로 무한루프를 도는 프로그램

Imp의 의미 - Small-step semantics

전이 시스템 (transition system)

상태들의 집합 S , 전이 관계 $\rightarrow \subseteq S \times S$.

$(p, q) \in \rightarrow$ 는 $p \rightarrow q$ 와 같이 표기한다.

Imp의 의미 - Small-step semantics

전이 시스템 (transition system)

상태들의 집합 S , 전이 관계 $\rightarrow \subseteq S \times S$.

$(p, q) \in \rightarrow$ 는 $p \rightarrow q$ 와 같이 표기한다.

Imp의 의미를 $S = (\text{Stmt} \uplus \{\cdot\}) \times \text{Mem}$ 을 상태로 가지는 전이 시스템으로 정의할 수 있다.

$$\mathcal{P}(S \times S)$$

$\text{Stmt} \uplus \{\cdot\}$ 는 프로그램 카운터와 비슷한 역할

Imp의 의미 - Small-step semantics

$$S = x \leftarrow 1; y \leftarrow 2; z \leftarrow x + y$$

Imp의 의미 - Small-step semantics

$$S = x \leftarrow 1; y \leftarrow 2; z \leftarrow x + y$$

$\{ * \mapsto 0 \}$ 에서 S 를 실행시키는 경우

$$(x \leftarrow 1; y \leftarrow 2; z \leftarrow x + y, \{ * \mapsto 0 \})$$



$$(y \leftarrow 2; z \leftarrow x + y, \{ x \mapsto 1, * \mapsto 0 \})$$



$$(z \leftarrow x + y, \{ y \mapsto 2, x \mapsto 1, * \mapsto 0 \})$$



$$(\cdot, \{ z \mapsto 3, y \mapsto 2, x \mapsto 1, * \mapsto 0 \})$$

Imp의 의미 - Small-step semantics

현실의 프로그램들은 외부 세계와 상호작용(키보드 입력, 화면 출력, 디스크 작업, ...)을 하는데, 이것을 어떻게 표현할 수 있을까?

Imp의 의미 - Small-step semantics

현실의 프로그램들은 외부 세계와 상호작용(키보드 입력, 화면 출력, 디스크 작업, ...)을 하는데, 이것을 어떻게 표현할 수 있을까?

Imp를 문법을 다음과 같이 확장시켜보자.

$$\text{Stmt } S ::= \text{skip} \mid x \leftarrow E \mid S_1; S_2 \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \mid \text{while } B \text{ do } S \\ \text{put } E \mid x \leftarrow \text{get}$$

Imp의 의미 - Small-step semantics

라벨이 있는 전이 시스템 (labeled transition system)

상태들의 집합 S , 라벨들의 집합 L , 전이 관계 $\rightarrow \subseteq S \times L \times S$

$$p \xrightarrow{e} q \quad \text{where } p, q \in S \text{ and } e \in L$$

“상태 p 에서 이벤트 e 를 발생시키며 상태 q 로 전이할 수 있다”

Imp의 의미 - Small-step semantics

라벨이 있는 전이 시스템 (labeled transition system)

상태들의 집합 S , 라벨들의 집합 L , 전이 관계 $\rightarrow \subseteq S \times L \times S$

$$p \xrightarrow{e} q \quad \text{where } p, q \in S \text{ and } e \in L$$

“상태 p 에서 이벤트 e 를 발생시키며 상태 q 로 전이할 수 있다”

$$\mathcal{P}(S \times L \times S)$$

$$S = (\text{Stmt} \uplus \{\cdot\}) \times \text{Mem}$$

$$L = \{\text{put } n \mid n \in \mathbb{Z}\} \cup \{\text{get } n \mid n \in \mathbb{Z}\}$$

Imp의 의미 - Small-step semantics

라벨이 있는 전이 시스템 (labeled transition system)

상태들의 집합 S , 라벨들의 집합 L , 전이 관계 $\rightarrow \subseteq S \times L \times S$

$$p \xrightarrow{e} q \quad \text{where } p, q \in S \text{ and } e \in L$$

“상태 p 에서 이벤트 e 를 발생시키며 상태 q 로 전이할 수 있다”

$$\mathcal{P}(S \times L \times S)$$

$$S = (\text{Stmt} \uplus \{\cdot\}) \times \text{Mem}$$

$$L = \{\text{put } n \mid n \in \mathbb{Z}\} \cup \{\text{get } n \mid n \in \mathbb{Z}\}$$

이벤트는 프로그램과 외부세계의 상호작용을 표현한다. 외부에서 관측 되는것이 이벤트 뿐이라면, 상태를 배제하고 이벤트만으로 프로그램의 의미를 정의할 수 있지 않을까?

Imp의 의미 - 트레이스, 행동

트레이스 (trace)

프로그램의 한 실행에서 발생한 관측 가능한 이벤트들의 시퀀스.
프로그램의 행동 beh는 발생 가능한 모든 트레이스들의 집합.

$$\text{beh} : \text{Stmt} \rightarrow \mathcal{P}(\text{Trace}(L))$$

Imp의 의미 - 트레이스, 행동

트레이스 (trace)

프로그램의 한 실행에서 발생한 관측 가능한 이벤트들의 시퀀스.
프로그램의 행동 beh는 발생 가능한 모든 트레이스들의 집합.

$$\text{beh} : \text{Stmt} \rightarrow \mathcal{P}(\text{Trace}(L))$$

Trace는 종료하지 않는 프로그램의 트레이스를 표현할 수 있도록 주의 깊게 정의 되어야 한다. CompCert에서는 프로그램의 트레이스를 다음과 같이 4가지로 분류한다.

- ▶ Termination, 유한개의 관측 가능한 이벤트를 발생시킨 후 종료한 경우.
- ▶ Divergence, 유한개의 관측 가능한 이벤트를 발생시킨 후 무한히 실행되며 종료하지 않는 경우.
- ▶ Reactive divergence, 무한개의 관측 가능한 이벤트를 지속적으로 발생시키는 경우.
- ▶ Going wrong, 유한개의 관측 가능한 이벤트를 발생시킨 후 비정상적으로 종료한 경우.