좁게 어림잡는 정적 분석기의 설계

(Design of an Under-Approximate Static Analyzer)

지도교수 : 이광근

이 논문을 공학학사 학위 논문으로 제출함.

2025년 12월 17일

서울대학교 공과대학

컴퓨터공학부

권민정

2026년 2월

좁게 어림잡는 정적 분석기의 설계

(Design of an Under-Approximate Static Analyzer)

지도교수 : 이광근

이 논문을 공학학사 학위 논문으로 제출함.

2025년 12월 17일

서울대학교 공과대학

컴퓨터공학부

권민정

2026년 2월

국문초록

정적 분석은 프로그램을 실제로 실행하지 않고도 잠재적인 오류를 탐지하기 위한 기법으로, 전통적으로는 프로그램의 모든 가능한 실행을 포함하도록 넓게 어림잡는*over-approximation* 방식의 분석이 주로 사용되어 왔다. 이러한 분석은 오류가 발생하지 않음을 보장하는 데에는 효과적이지만, 분석 결과에 포함된 오류가 실제 실행에서도 반드시 발생한다고 보장하지는 못한다는 한계를 가진다. 이로 인해 분석 결과에 다수의 거짓 경보가 포함될 수 있으며, 이는 분석 결과의 활용도를 저하시킨다.

본 논문은 이러한 한계를 보완하기 위해, 프로그램의 실제 실행 중 일부만을 포함하되 분석 결과에 포함된 실행은 모두 실제로 발생함을 보장하는 좁게 어림잡는*under-approximation* 정적 분석에 주목한다. 특히, 기존의 요약 해석*abstract interpretation* 틀에서 널리 사용되는 의미 함수 기반의 분석 구조를 바탕으로, 좁게 어림잡는 분석기를 체계적으로 설계하기 위한 일반적인 분석 틀을 제시한다.

먼저 작은 보폭 의미구조와 모둠 의미구조를 이용해 프로그램의 실행 의미를 정의하고, 이를 요약한 넓게 어림잡는 기존의 분석 틀을 정리한다. 이후 이와 구조적으로 유사하되 포함 관계의 방향을 반대로 갖는 좁게 어림잡는 요약 의미구조를 정의하고, 초기 상태, 요약 의미 함수, 요약 뭉침 연산자가 만족해야 할 조건들을 제시한다. 이러한 조건 하에서, 제안한 분석 틀에 따라 계산된 분석 결과는 실제 실행 의미에 포함됨을 증명하며, 반복 계산을 어느 시점에서 중단하더라도 그 시점까지의 결과가 여전히 올바른 분석 결과가 됨을 보인다.

또한, 제안한 분석 틀을 간단한 명령형 언어에 적용하여 좁게 어림잡는 분석기를 설계하고, 분석이 잘 이루어지는 사례와 잘 이루어지지 않는 사례를 통해 좁게 어림잡는 분석의 특성과 한계를 살펴본다. 이를 통해 좁게 어림잡는 분석에서는 요약 도메인의 표현력이 분석의 성능과 정확성에 중요한 영향을 미친다는 점을 확인하고, 넓게 어림잡는 분석에서 널리 사용되는 구간*interval* 도메인이 갖는 한계를 논의한다.

본 논문은 좁게 어림잡는 정적 분석을 의미 함수 기반의 틀에서 정식화함으로써, 실제 오류의 존재를 보장하는 분석 결과를 이론적으로 뒷받침할 수 있는 기반을 제시한다. 이를 통해 넓게 어림잡는 분석과 좁게 어림잡는 분석을 상호 보완적으로 활용하기 위한 이론적 토대를 마련하고, 향후 보다 정밀한 요약 도메인 및 분석 기법에 대한 연구 방향을 제시한다.

주요어: 정적 분석, 프로그램 분석, 요약 해석, 좁게 어림잡는 분석, 완전성

목 차

표 목차

그림 목차

제 1 장 서론

요약해석*abstract interpretation* 기반의 정적 분석*static analysis*은 프로그램의 실행을 어림잡아 모델링함으로써 프로그램의 잠재적인 오류 가능성을 미리 탐지한다. 이 때 프로그램의 실제 실행 행동을 모두 포섭하되 실제보다 더 포함하여 넓게 어림잡는 방식*over-approximation*으로 프로그램 실행을 어림잡는다. 넓게 어림잡는 분석에서 오류가 발견되지 않는다면, 실제 실행에서도 해당 오류가 절대 발생하지 않음을 보장할 수 있다. 반대로, 오류가 발견되었다고 해서 실제 실행에서도 반드시 그 오류가 발생한다고 말할 수는 없다. 넓게 어림잡는 분석의 이러한 한계를 극복하기 위해, 프로그램의 실제 실행 행동의 일부만을 포함하되 놓치는 실행이 있을 수 있도록 좁게 어림잡는 방식*under-approximation*의 분석을 함께 사용하는 것을 고려할 수 있다.

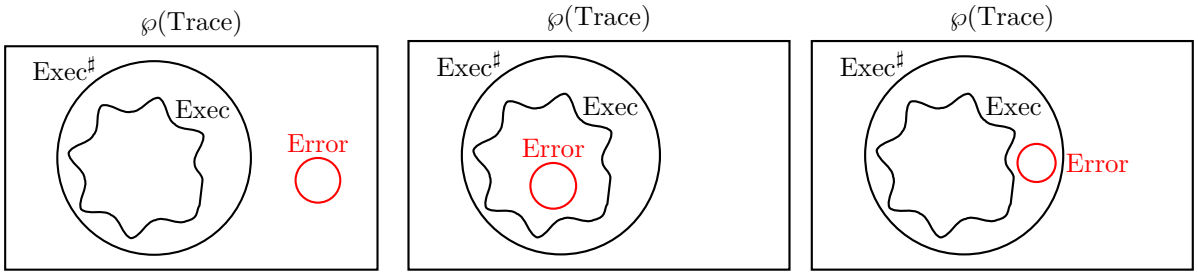
제 1 절 연구 배경 및 동기

먼저 프로그램의 정확한 실행 의미*semantics*는 그 프로그램의 가능한 실행을 모두 모은 집합이라고 할 수 있다. 프로그램의 실행에서 관찰 가능한 것은 메모리 상태의 변화인데, 실행 단계별로 메모리 상태의 변화를 모은 것을 실행 발자국*trace*이라고 한다. 넓게 어림잡는 분석에서는, 프로그램의 실제 실행 집합을 $\mathrm{Exec} \in \wp(\mathrm{Trace})$, 분석 결과를 $\mathrm{Exec}^\sharp \in \wp(\mathrm{Trace})$라 할 때

$$\mathrm{Exec} \subseteq \mathrm{Exec}^\sharp$$

와 같은 포함 관계가 성립하도록 설계한다. 이 조건 덕분에, Fig. 1.1a와 같이 분석 결과인 $\mathrm{Exec}^\sharp$가 어떤 오류 상황을 전혀 포함하지 않는다면, 실제 실행에서도 해당 오류가 절대 발생하지 않음을 보장할 수 있다. 그러나 반대로 분석 결과가 오류를 포함한다고 해도 실제 실행에서도 반드시 그 오류가 발생한다고 말할 수는 없다.

상용 정적 분석기의 결과는 흔히 경보*alarm*의 형태로 제시된다. 분석기는 프로그램의 특정 위치에서 잠재적인 오류 가능성을 발견하면 해당 위치에 경보를 달고, 사용자는 이 경보가 실제 오류 때문에 발생했는지(true alarm, Fig. 1.1b), 또는 분석이 더 잡아 근사하기 때문에 발생한 거짓 경보인지(false alarm, Fig. 1.1c) 직접 판별해야 한다. 실제 시스템에서는 경보가 매우 많이 발생하며, 이 중 상당수가 거짓 경보인 경우가 많다. 이로 인해 분석 결과를 해석하고 선별하는 비용이 커지고, 사용자는 분석기의 경고를 신뢰하기



(a) 경보 없음 (b) 실제 경보 (c) 거짓 경보

Figure 1.1: 넓게 어림잡는 분석에서의 경보

어려워진다.

이러한 맥락에서 넓게 어림잡는 분석으로만은 충분하지 않은 상황이 분명히 존재한다. 이때 실제 실행에 포함되는 결과를 내놓도록 좁게 어림잡음으로써 "이러한 실행이 정말 존재한다"고 말해줄 수 있는 분석이 함께 있다면, 경보 중 일부는 실제로 오류가 발생한다는 정보를 함께 제공할 수 있을 것이다.

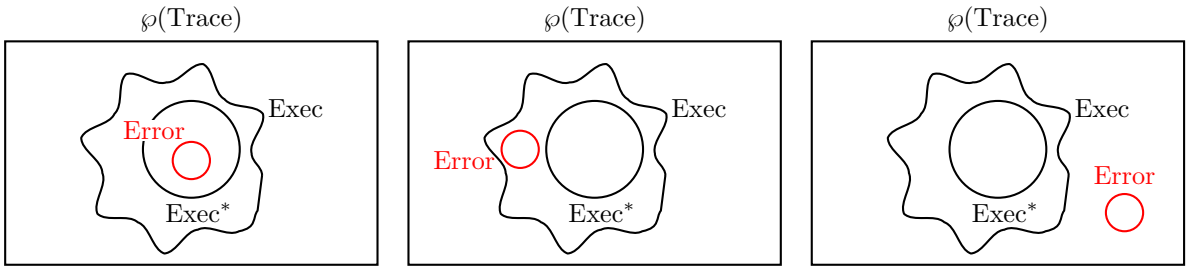
제 2 절 좁게 어림잡는 분석

좁게 어림잡는 분석은 넓게 어림잡는 분석과 반대 방향의 포함 관계를 갖는다. 좁게 어림잡는 분석기의 결과를 Exec^* 라 하면,

$$\mathrm{Exec}^* \subseteq \mathrm{Exec}$$

와 같은 관계가 성립하도록 설계한다. 즉, 분석기가 보고하는 결과는 실제 실행에서 실제로 발생할 수 있는 경우들로만 "가짜 없이" 이루어지도록 보장한다. 이러한 분석 결과는 프로그램에 오류가 전혀 없다는 것을 보이는 안전성 증명에는 적합하지 않지만, 반대로 "이러한 실행이 실제로 존재한다"는 것을 보이는 데 적합하다.

이를 분석기에서의 경보와 결합해 생각할 수 있다. 어떤 프로그램 위치에 대해 넓게 어림잡는 분석이 오류 가능성을 보고해 경보를 달았다고 하자. 이때 동일한 위치를 대상으로 좁게 어림잡는 분석을 수행한



(a) 오류 존재, 실제 경보 (b) 오류 존재, 경보 없음 (c) 오류 없음, 경보 없음

Figure 1.2: 좁게 어림잡는 분석에서의 경보

Table 1.1: 넓게 어림잡는 분석과 좁게 어림잡는 분석에서 경보 조합의 의미

넓은 어림	좁은 어림	의미
🔔	🔔	실제 경보
🔔	🔕	실제 또는 거짓 경보
🔕	🔕	오류가 없음

결과 오류를 실제로 발생시키는 실행이 Exec^* 안에 포함되어 있다면, 우리는 이 경보가 실제 경보라는 것을 확신할 수 있다 (Fig. 1.2a). 반대로 좁게 어림잡는 분석 결과가 해당 오류를 포함하지 않는다고 하더라도, 그것만으로 오류가 없다고 단정할 수는 없다 (Figs. 1.2b와 1.2c).

요약하면, 넓게 어림잡는 분석은 "이런 오류는 절대 발생하지 않는다"는 주장을 뒷받침하는 데 적합한 반면, 좁게 어림잡는 분석은 "이런 오류가 실제로 발생하는 실행이 존재한다"는 주장을 뒷받침하는 데 적합하다.

Table 1.1에서 넓은 어림은 $\mathrm{Exec} \subseteq \mathrm{Exec}^\sharp$를 만족하는 넓게 어림잡는 분석의 결과 $\mathrm{Exec}^\sharp$가 오류를 포함하는지를 나타내고, 좁은 어림은 $\mathrm{Exec}^* \subseteq \mathrm{Exec}$를 만족하는 좁게 어림잡는 분석의 결과 Exec^*가 오류를 포함하는지를 나타낸다. 두 종류의 분석을 적절히 결합하면, 사용자는 경보에 대해 Table 1.1와 같은 판단을 할 수 있다.

이 논문은 오류가 실제로 존재한다는 것을 보여주는 좁게 어림잡는 분석기를 체계적으로 설계하고 그 성질을 증명할 수 있는 일반적인 틀*framework*을 제시하는 것을 목표로 한다.

제 3 절 연구의 목표와 기여

이 논문의 궁극적인 목표는, 실행발자국 의미구조*transitional semantics*를 바탕으로 한 기존 요약 해석 틀과 비슷한 형태로 좁게 어림잡는 분석기를 구성하기 위한 일반적인 분석 틀을 제시하고, 이 틀이 제공하는 분석 결과의 성질을 이론적으로 보장하는 것이다.

보다 구체적으로, 이 논문의 기여를 다음과 같이 요약할 수 있다.

1. **좁게 어림잡는 분석기 설계를 위한 틀 제시**

   프로그램의 의미구조*concrete semantics*를 의미 함수*semantic function* F로 표현하고, 이를 좁게 어림잡는 요약 의미함수*under-approximate abstract semantic function* F^*와 뭉침 요약 연산자 $\sqcup^*$를 정의할 때, 이들이 만족해야 할 일반적인 조건들을 제시한다. 또한 이러한 조건 하에서, F^*의 반복 적용과 $\sqcup^*$ 연산에 의해 얻어지는 결과가 실제 실행 의미에서 모두 실제로 나타난다는 것을 증명한다.

2. **제안한 틀을 이용한 분석기 예시와 어려움에 대한 고찰**

   간단한 언어를 정의하고, 그 언어의 의미구조와 좁게 근사하는 요약 공간(A^*)을 설계한 뒤, 앞에서 제시한 분석 틀에 맞는 F^*와 $\sqcup^*$을 정의하고, 예시 프로그램에 대한 동작을 확인한다. 이를 통해 실제

로 의미 있는 좁게 어림잡는 분석기를 만드는 과정에서 어떤 어려움이 있는지 살펴본다. 그리고 분석 결과가 유용해지기 위해 모둠 의미구조*concrete collecting semantics*와 요약 공간*abstract domain*을 어떻게 설계해야 하는지에 대해 논의한다.

제 2 장 배경

이 장에서는 기존 넓게 어림잡는 분석 요약 해석 틀을 정리한다. 작은 보폭*small-step* 의미구조를 바탕으로 한 모둠 의미구조를 정의하고, 이를 요약하는 요약 의미구조*abstract semantics*를 정의한다. 본 장에서 정의하는 넓게 어림잡는 분석은 Cousot와 Cousot이 제안한 요약 해석 이론에 기반하고 [1], [2], 구체적인 분석 틀의 구조는 [3]의 서술 방식을 따른다.

제 1 절 모둠 의미 구조

1.1 프로그램 상태와 의미공간 D

프로그램은 명령어 라벨*label* 집합 $\mathbb{L}$과 메모리 상태 집합 $\mathbb{M}$으로 구성된다고 가정한다. 상태는 현재 실행 중인 명령어 라벨과 메모리 상태의 쌍으로 표현한다.

$$\mathbb{S} = \mathbb{L} \times \mathbb{M}.$$

정적 분석에서 관심 있는 것은 하나의 상태가 아니라, 도달 가능한 상태들의 집합이다. 따라서 모둠 의미구조의 의미공간*concrete domain* D는 상태 집합의 집합

$$D = \wp(\mathbb{S})$$

로 두고, 부분순서는 포함 관계 $\subseteq$로 정의한다. 두 집합의 join은 합집합으로 주어진다.

$$X \sqcup Y = X \cup Y.$$

1.2 작은 보폭 의미구조와 Step 연산

프로그램의 한 단계 실행 단계를 나타내는 작은 보폭 의미구조는 상태 사이의 전이 관계

$$\hookrightarrow \subseteq \mathbb{S} \times \mathbb{S}$$

로 주어진다고 가정한다. 예를 들어 $(\ell, m) \hookrightarrow (\ell', m')$는 라벨 ℓ에서 메모리 상태 m에 대해 명령을 실행한 결과 라벨 ℓ'과 메모리 상태 m'에 도달했음을 뜻한다.

한 개의 상태에 대해 정의된 전이 관계를 상태 여러 개의 집합에 대해 사용하기 위해 다음과 같이 Step 함수를 정의한다.

$$\mathrm{Step} : \wp(\mathbb{S}) \to \wp(\mathbb{S})$$
$$\mathrm{Step}(X) = \{\, s' \in \mathbb{S} \mid s \hookrightarrow s',\ s \in X \,\}.$$

즉, $\mathrm{Step}(X)$는 X에 속한 상태들에서 한 번 작은 보폭 실행할 때 도달할 수 있는 모든 상태의 집합이다.

1.3 의미함수 F와 모듬 의미구조

프로그램의 초기 상태 집합을 $I \in \wp(\mathbb{S})$라고 하자. 이때 프로그램의 모듬 의미구조를 정의하는 의미함수를 다음과 같이 정의한다.

$$F : \wp(\mathbb{S}) \to \wp(\mathbb{S})$$
$$F(X) = I \cup \mathrm{Step}(X).$$

이 F를 반복적으로 적용하면, 프로그램 실행 동안 도달 가능한 상태 집합이 점차 늘어난다. 이를 이용하여 I에서 도달 가능한 모든 상태들의 집합, 즉 프로그램의 모듬 의미구조를 다음과 같이 정의할 수 있다.

$$\mathrm{Sem} = \mathrm{lfp}\ F$$

제 2 절 요약 실행발자국 의미구조

이 절에서는 위에서 정의한 모듬 의미구조 $F : \wp(\mathbb{S}) \to \wp(\mathbb{S})$를 더 잡아 근사하는 요약 의미구조 $F^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp$를 정의한다. 이를 위해 의미공간 $\wp(\mathbb{S})$와 요약 공간 $\mathbb{S}^\sharp$ 사이에 갈로아 연결이 있다고 가정하고, 두 의미함수 F와 $F^\sharp$가 만족해야 하는 안전성 조건을 정리한다.

2.1 요약 공간

먼저, 모듬 의미구조에서 사용하는 메모리 집합 도메인 $\wp(\mathbb{M})$과 이를 요약하는 공간 $\mathbb{M}^\sharp$를 정의한다. 이때, 요약 공간 $\mathbb{M}^\sharp$는 CPO이어야 하고 두 공간은 갈로아 연결되어 있어야 한다.

$$(\wp(\mathbb{M}), \subseteq) \underset{\alpha_M}{\overset{\gamma_M}{\leftrightarrows}} (\mathbb{M}^\sharp, \sqsubseteq_M)$$

위 조건을 만족하도록 $\mathbb{M}^\sharp$을 잡으면, 상태 집합 도메인 $\wp(\mathbb{S})$에 대한 요약 공간 $\mathbb{S}^\sharp$를 다음과 같이 정의할 수 있고, 이 또한 CPO이며 다음의 갈로아 연결을 만족한다.

$$\mathbb{S}^\sharp = \mathbb{L} \to \mathbb{M}^\sharp$$

$$(\wp(\mathbb{S}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{S}^\sharp, \sqsubseteq)$$

2.2 요약 의미구조

이제 위에서 정의한 모둠 의미구조를 요약하여 유한하게 계산 가능한 요약 의미함수 $F^\sharp$를 다음과 같이 정의한다. 요약 전이관계 $\hookrightarrow^\sharp$와 요약 뭉침 연산 $\sqcup^\sharp$을 정의하면 아래와 같이 $F^\sharp$를 정의할 수 있다.

모둠 의미구조	**넓게 어림잡는 요약 의미구조**
$\mathbb{S} = \mathbb{L} \times \mathbb{M}$	$\mathbb{S}^\sharp = \mathbb{L} \to \mathbb{M}^\sharp$
$F : \wp(\mathbb{S}) \to \wp(\mathbb{S})$	$F^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp$
$F(X) = I \cup Step(X)$	$F^\sharp(X^\sharp) = \alpha(I) \cup^\sharp Step^\sharp(X^\sharp)$
$Step = \breve{\wp}(\hookrightarrow)$	$Step^\sharp = \wp(\mathrm{id}, \cup^\sharp_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp)$
$\hookrightarrow \subseteq (\mathbb{L} \times \mathbb{M}) \times (\mathbb{L} \times \mathbb{M})$	$\hookrightarrow^\sharp \subseteq (\mathbb{L} \times \mathbb{M}^\sharp) \times (\mathbb{L} \times \mathbb{M}^\sharp)$

제 3 절 안전성 조건

이제 요약 의미구조가 실제 의미구조를 놓침없이 모두 포섭하는 안전한*sound* 분석이 되기 위한 조건들을 정리한다.

1. 갈로아 연결 메모리 집합 공간 $\wp(\mathbb{M})$과 이를 요약하는 공간 $\mathbb{M}^\sharp$, 그리고 상태 집합 도메인 $\wp(\mathbb{S})$과 이를 요약하는 요약 공간 $\mathbb{S}^\sharp$ 사이에는 각각 다음과 같은 갈로아 연결이 주어져 있어야 한다.

$$(\wp(\mathbb{M}), \subseteq) \underset{\alpha_M}{\overset{\gamma_M}{\leftrightarrows}} (\mathbb{M}^\sharp, \sqsubseteq_M)$$
$$(\wp(\mathbb{S}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{S}^\sharp, \sqsubseteq)$$

2. 요약 한걸음 연산자*abstract one-step transition*, 요약 뭉침 연산자*abstract join* $\sqcup^\sharp$의 안전성 Concrete semantics F를 abstract semantics $F^\sharp$가 안전하게 포섭하려면 $\hookrightarrow, \hookrightarrow^\sharp$ 그리고 $\cup, \sqcup^\sharp$가 다음 조건을 만족하

면 충분하다.

$$\breve{\wp}(\hookrightarrow) \circ \gamma \subseteq \gamma \circ \breve{\wp}(\hookrightarrow^\sharp)$$

$$\cup \circ (\gamma\_, \gamma\_) \subseteq \gamma\_ \circ \sqcup^\sharp\_$$

위의 두 조건을 만족하도록 분석을 설계하면 안전하게 놓침없이 따지는 분석을 얻을 수 있다. 즉,

$$\mathrm{lfp}\, F \subseteq \gamma(\mathrm{lfp}\, F^\sharp)$$

가 성립함을 보장할 수 있다.

제 3 장 좁게 어림잡는 분석 틀

이 장에서는 [3]의 넓게 어림잡는 요약 해석 틀의 구조를 유지한 채, 포함 관계를 반대로 두는 방식으로 좁게 어림잡는 분석 틀을 제시한다. 기준이 되는 프로그램의 실행의미인 모듬 의미구조는 2장에서 정의한 것과 동일하게 사용한다. 이를 바탕으로, 좁게 어림잡는 요약 의미함수 F^*와 요약 뭉침 연산자 $\sqcup^*$를 정의하고, 이들이 만족해야 할 조건들을 제시한다. 마지막으로, 이러한 조건들을 만족하도록 설계한 분석기의 실행 결과는 실제에서 모두 나타난다는 것을 증명한다.

제 1 절 좁게 어림잡는 요약 의미구조

1.1 좁게 어림잡는 요약 공간

넓게 어림잡는 분석에서 $\mathbb{S}^\sharp$을 도입한 것과 마찬가지로, 프로그램의 실행을 요약해서 다루기 위한 요약 공간 $\mathbb{S}^*$를 도입한다. 이 요약 공간의 원소는 "요약된 실행 집합"을 나타낸다고 생각할 수 있고, 이 집합에 있는 모든 실행 상태는 실제 실행에서 반드시 나타난다.

$\mathbb{S}^*$은 의미 공간 $\wp(\mathbb{S})$로의 구체화 함수가 있어야 한다.

$$\gamma^* : \mathbb{S}^* \to \wp(\mathbb{S})$$

$a^* \in \mathbb{S}^*$이 어떤 프로그램의 요약 실행 의미라면, $\gamma^*(a^*) \in \wp(\mathbb{S})$는 실제 실행에서 모두 나타난다. 이를 만족하기 위해 $\gamma^*(\bot^*) = \bot$이어야 한다. 이 장의 나머지 부분에서는 편의상 γ^*를 간단히 γ라고 쓴다. 여기서 사용하는 γ는 갈로아 연결의 구체화 함수가 아니라, 단순히 요약 공간의 원소를 실제 실행 집합으로 변환하는 함수임에 유의하자.

넓게 어림잡는 분석에서는 $\mathbb{S}^\sharp$이 CPO이며, 요약 공간 $\wp(\mathbb{S})$과 Galois 연결을 이루도록 요구했지만, 좁게 어림잡는 분석에서는 이를 요구하지 않는다.

1.2 좁게 어림잡는 요약 의미구조

넓게 어림잡는 분석에서 $\hookrightarrow^\sharp$와 $\sqcup^\sharp$을 정의하여 $F^\sharp$를 구성한 것과 마찬가지로, 좁게 어림잡는 분석에서는 $\hookrightarrow^*$와 $\sqcup^*$을 사용하여 F^*를 구성한다.

모둠 의미구조	좁게 어림잡는 요약 의미구조
$\mathbb{S} = \mathbb{L} \times \mathbb{M}$	$\mathbb{S}^* = \mathbb{L} \to \mathbb{M}^*$
$F : \wp(\mathbb{S}) \to \wp(\mathbb{S})$	$F^* : \mathbb{S}^* \to \mathbb{S}^*$
$F(X) = I \cup Step(X)$	$F^*(X^*) = I^* \sqcup^* Step^*(X^*)$
$Step = \breve{\wp}(\hookrightarrow)$	$Step^* = \wp(\mathrm{id}, \sqcup_M^*) \circ \pi \circ \breve{\wp}(\hookrightarrow^*)$
$\hookrightarrow \subseteq (\mathbb{L} \times \mathbb{M}) \times (\mathbb{L} \times \mathbb{M})$	$\hookrightarrow^* \subseteq (\mathbb{L} \times \mathbb{M}^*) \times (\mathbb{L} \times \mathbb{M}^*)$

제 2 절 완전성 조건

본 논문에서의 완전성이란 분석 결과가 실제 실행에 포함됨을 의미한다. 올바르게 좁게 어림잡는 분석기를 설계하기 위해서는 초기 상태, 요약 뭉침 연산자, 의미함수에 대해 각각 특정한 조건을 요구한다. 다음에서는 이러한 조건들과, 조건을 만족한 경우에 분석기가 실제 실행의 완전한 요약이 됨을 증명한다.

Definition 1 (좁게 어림잡는 초기 상태)**.** I^*가 I의 좁게 어림잡는 초기 상태라 함은

$$\gamma(I^*) \subseteq I$$

이 성립하는 것을 말한다.

Definition 2 (좁게 어림잡는 요약 뭉침 연산자)**.** 연산자 $\sqcup^* : \mathbb{S}^* \times \mathbb{S}^* \to \mathbb{S}^*$가 좁게 어림잡는 요약 뭉침 연산자라 함은, 모든 $a_0^*, a_1^* \in \mathbb{S}^*$에 대해

$$\gamma(a_0^* \sqcup^* a_1^*) \subseteq \gamma(a_0^*) \cup \gamma(a_1^*)$$

이 성립하는 것을 말한다.

Definition 3 (좁게 어림잡는 요약 의미 함수)**.** 함수 $F^* : \mathbb{S}^* \to \mathbb{S}^*$가 좁게 어림잡는 요약 의미 함수라 함은, 모든 $a^* \in \mathbb{S}^*$에 대해

$$\gamma(F^*(a^*)) \subseteq F(\gamma(a^*))$$

이 성립하는 것을 말한다.

제 3 절 완전성과 언제 멈춰도 괜찮은 성질

이제 위의 조건에 따라 설계한 분석은 실제 실행에서 모두 관찰할 수 있다는 것을 증명한다. 분석 결과는 F^*를 반복 적용하고 얻은 정보를 $\sqcup^*$ 연산으로 합쳐서 얻어진다.

$$\text{분석 결과} = \bigsqcup_{n\geq 0}^{*} F^{*n}(\bot)$$

실제 실행 의미는 $\text{lfp}\, F$로 주어지며, 이는 다음과 같이 표현할 수 있다.

$$Sem = \text{lfp}\, F = \bigcup_{n\geq 0} F^n(\bot).$$

먼저, 아래의 보조 정리를 증명한다.

Lemma 1 (n-단계 좁게 어림잡기의 완전성)**.** *Theorem 1*의 조건들 하에서, 임의의 $n \geq 0$에 대해

$$\gamma(F^{*n}(\bot^*)) \subseteq F^n(\bot).$$

을 만족한다.

Proof. n에 대한 귀납으로 보인다.

- $n = 0$
  $\gamma(F^{*0}(\bot^*)) = \gamma(\bot^*) = \bot$이므로 자명하게 $\gamma(F^{*0}(\bot^*)) \subseteq F^0(\bot)$이 성립한다.
- $n \to n+1$

$$\begin{aligned}
\gamma(F^{*(n+1)}(\bot^*)) &= \gamma(F^*(F^{*n}(\bot^*))) \\
&\subseteq F(\gamma(F^{*n}(\bot^*))) && \because \text{좁게 어림잡는 요약 의미 함수의 정의} \\
&\subseteq F(F^n(\bot)) && \because \text{귀납 가정, 단조성 } F \\
&= F^{n+1}(\bot).
\end{aligned}$$

따라서 모든 $n \geq 0$에 대해 $\gamma(F^{*n}(\bot^*)) \subseteq F^n(\bot)$이 성립한다. □

보조정리를 이용하여 좁게 어림잡는 분석기의 결과가 실제 실행 의미에 포함된다는 것을 증명한다.

Theorem 1 (좁게 어림잡기의 완전성)**.** 만약 $I^*, \sqcup^*$ 와 F^* 가 각각 *Definition 1*, *Definition 2*, *Definition 3*에서 정의한 좁게 어림잡는 초기 상태, 요약 뭉침 연산자, 요약 의미 함수라면,

$$\gamma\left(\bigsqcup_{n\geq 0}^{*} F^{*n}(\bot^*)\right) \subseteq \mathrm{lfp}\, F.$$

을 만족하여 좁게 어림잡는 분석기의 결과는 실제 실행 의미에 포함된다.

Proof.

$$\begin{aligned}
\gamma\left(\bigsqcup_{n\geq 0}^{*} F^{*n}(\bot^*)\right) &\subseteq \bigcup_{n\geq 0} \gamma(F^{*n}(\bot^*)) && \because \text{좁게 어림잡는 요약 뭉침 연산자의 정의} \\
&\subseteq \bigcup_{n\geq 0} F^n(\bot) && \because \textit{Lemma } 1 \\
&= \mathrm{lfp}\, F.
\end{aligned}$$

□

넓게 어림잡는 분석기에서는 분석이 종료되는 것을 보장하기 위해 유한 높이의 요약 공간을 사용하거나 축지법*widening* 연산자를 도입하는 등의 기법을 사용한다. 반면, 본 절에서 정의한 좁게 어림잡는 분석기는 이러한 기법을 사용하지 않으므로, Theorem 1에 나타난 $\gamma\left(\bigsqcup_{n\geq 0}{}^{*} F^{*n}(\bot^*)\right)$의 계산이 끝나지 않을 수도 있다. 따라서 실제 구현에서는 유한한 시간 내에 분석을 끝내기 위해 반복을 적절한 시점에서 중단해야 하며, 이때 중단된 시점의 결과가 여전히 의미 함수에 포함된다는 것을 보장해야 한다.

Dataflow analysis 분야에서는 이와 같이 모든 중간 단계의 결과를 사용할 수 있는 분석을 *pessimistic analysis*, 반대로 고정점에 도달해야만 의미 있는 결과인 분석을 *optimistic analysis*라고 부른다. 앞 장의 넓게 어림잡는 분석은 optimistic analysis에 해당하므로 분석의 종료를 따로 보장해야 하지만, 본 절의 좁게 어림잡는 분석은 pessimistic analysis에 해당하므로 반복을 어느 지점에서 중단하더라도 그 시점까지의 결과가 올바른 좁게 어림잡는 분석 결과가 됨을 자연스럽게 보장한다.

좁게 어림잡는 분석기는 반복을 어느 시점에서 멈추더라도 그 시점까지의 결과가 실제 실행 의미의 좁게 어림잡는 분석 결과가 됨을 보장하는, 즉 위 분석기는 *pessimistic analysis*에 해당된다는 내용의 다음 정리를 증명한다.

Corollary 1 (언제 멈춰도 괜찮은 분석)**.** Theorem 1의 조건들 하에서, 임의의 유한 단계 n에 대해서 다음이 성립한다.

$$\gamma\left(\bigsqcup_{k\leq n}^{*} F^{*k}(\bot^*)\right) \subseteq \mathrm{lfp}\, F.$$

Proof.

$$\gamma\left(\bigsqcup_{k\le n}^{*} F^{*k}(\bot^*)\right) \subseteq \bigcup_{k\le n} \gamma(F^{*k}(\bot^*)) \qquad \because \text{좁게 어림잡는 요약 뭉침 연산자의 정의}$$

$$\subseteq \bigcup_{k\le n} F^k(\bot) \qquad \because \textit{Lemma } 1$$

$$\subseteq \mathrm{lfp}\, F.$$

□

제 4 장 사례 연구

본 장에서는 앞서 제시한 좁게 어림잡는 분석 틀을 사용하여 간단한 언어에 대한 좁게 어림잡는 분석기를 설계하는 과정을 살펴본다. 이를 통해 실제로 의미 있는 좁게 어림잡는 분석기를 만드는 과정에서 어떤 어려움이 있는지 살펴보고, 유용한 분석 결과를 얻기 위해 실행 의미와 요약 공간을 어떻게 설계해야 하는지에 대해 논의한다.

제 1 절 예시 언어

1.1 문법

예시 언어의 문법을 다음과 같이 정의한다.

Expression	E	$::=$	n	$n \in \mathbb{Z}$
		$\mid$	$E \oplus E$	$\oplus \in \{+, -, *, <\}$
		$\mid$	x	variable
		$\mid$	`readInt`	input
Command	C	$::=$	`skip`	
		$\mid$	x `:=` E	
		$\mid$	$C; C$	
		$\mid$	`if` E `then` C `else` C	
		$\mid$	`while` E `do` C	

1.2 모듬 의미구조

이 절에서는 mini 언어에 대한 작은 보폭 의미구조를 정의한다. 메모리 상태는 변수에서 정수로의 함수로 정의한다.

$$\mathbb{M} = Var \to \mathbb{Z}.$$

앞서 상태를 명령어 라벨과 메모리 쌍 $(\ell, \mathbb{M})$으로 정의했지만, 명령어 라벨과 그에 해당하는 명령어를 함께 가지고 있는 것이 편리하므로, 상태를 다음과 같이 다시 정의한다.

$$\mathbb{S} = (\mathbb{L} \times \mathbb{C}) \times \mathbb{M}.$$

상태 $(C, m) \in \mathbb{S}$는 메모리 상태 m에서 명령 C를 실행 중임을 뜻한다. 앞서 2장에서와 마찬가지로, 프로그램의 한 걸음 실행은 전이 관계

$$\hookrightarrow \subseteq \mathbb{S} \times \mathbb{S}$$

으로 정의한다. 또한, 본 언어에서는 gogo 명령어가 없기 때문에 프로그램의 실행 흐름이 고정되어 있다. 다음 실행할 명령어를 알려주는 next 함수, if문과 while문에서 조건 만족 여부에 따라 다음 실행할 명령어를 알려주는 nextTrue, nextFalse 함수가 다음과 같이 존재한다.

$$\mathsf{next}, \mathsf{nextTrue}, \mathsf{nextFalse} : \mathbb{L} \to \wp(\mathbb{L} \times \mathbb{C})$$

표현식 계산 먼저 표현식은 메모리 상태에 대한 큰 보폭 의미구조로 다음과 같이 계산한다.

$$\frac{}{m \vdash n \Downarrow n}\ \text{E-Int} \qquad \frac{}{m \vdash x \Downarrow m(x)}\ \text{E-Var}$$

$$\frac{m \vdash E_1 \Downarrow n_1 \qquad m \vdash E_2 \Downarrow n_2 \qquad n = n_1 \oplus n_2}{m \vdash E_1 \oplus E_2 \Downarrow n}\ \text{E-BinOp} \qquad \frac{n \in \mathbb{Z}}{m \vdash \texttt{readInt} \Downarrow n}\ \text{E-ReadInt}$$

여기서 $\oplus \in \{+, -, *, <\}$이며, 비교 연산 $<$의 결과는 참일 경우 1, 거짓일 경우 0으로 정의한다.

명령어의 작은 보폭 의미구조 명령에 대한 작은 보폭 의미구조는 상태 사이의 전이 관계

$$\hookrightarrow \subseteq \mathbb{S} \times \mathbb{S}$$

로 정의되며, 전이 규칙은 다음과 같다.

S-Skip

$$\frac{}{(\langle \ell, \texttt{skip} \rangle,\ m) \hookrightarrow (\mathsf{next}(\ell),\ m)}$$

S-Assign

$$\frac{m \vdash E \Downarrow n}{(\langle \ell, x \texttt{:=} E \rangle,\ m) \hookrightarrow (\mathsf{next}(\ell),\ m[x \mapsto n])}$$

S-Seq

$$\frac{}{(\langle \ell, C_1; C_2 \rangle,\ m) \hookrightarrow (\mathsf{next}(\ell),\ m)}$$

S-IfTrue

$$\frac{m \vdash E \Downarrow n \quad n \neq 0}{(\langle \ell, \texttt{if}\ E\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \rangle,\ m) \hookrightarrow (\mathsf{nextTrue}(\ell),\ m)}$$

S-IfFalse

$$\frac{m \vdash E \Downarrow 0}{(\langle \ell, \texttt{if}\ E\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \rangle,\ m) \hookrightarrow (\mathsf{nextFalse}(\ell),\ m)}$$

S-WhileTrue

$$\frac{m \vdash E \Downarrow n \quad n \neq 0}{(\langle \ell, \texttt{while}\ E\ \texttt{do}\ C \rangle,\ m) \hookrightarrow (\mathsf{nextTrue}(\ell),\ m)}$$

S-WhileFalse

$$\frac{m \vdash E \Downarrow 0}{(\langle \ell, \texttt{while}\ E\ \texttt{do}\ C \rangle,\ m) \hookrightarrow (\mathsf{nextFalse}(\ell),\ m)}$$

위와 같이 정의된 $\hookrightarrow$를 2장에서 정의한 $Step : \wp(\mathbb{S}) \to \wp(\mathbb{S})$에 대입하면, mini 언어에 대한 모듬 의미구조를 $F(X) = I \cup Step(X)$의 형태로 얻을 수 있다.

제 2 절 좁게 어림잡는 분석기 정의

mini 언어에 대해 2장의 틀에 맞게 좁게 어림잡는 분석기를 정의한다.

2.1 좁게 어림잡는 요약 도메인

먼저, $\wp$(Value)를 요약하는 요약 도메인으로 Itv (interval domain)을 사용한다.

$$\text{Itv} = \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{+\infty\}, l \leq u\} \cup \{\bot\}$$

$[-\infty, +\infty]$는 편의상 $\top$으로 쓴다. 이를 확장하여 $\mathbb{M}^*$, $\mathbb{S}^*$를 다음과 같이 정의한다.

$$\mathbb{M}^* = Var \to \text{Itv}$$

$$\mathbb{S}^* = (\mathbb{L} \times \mathbb{C}) \to \mathbb{M}^*$$

특정 위치에서 좁게 어림잡은 요약 메모리가 $m^* \in \mathbb{M}^* = \{x \mapsto [1,2], y \mapsto [2,4]\}$ 라면, 실제 실행에서 x, y 의 순서쌍 $\gamma(m^*) = \{(1,2),(1,3),(1,4),(2,2),(2,3),(2,4)\}$ 를 Fig. 4.1와 같이 모두 관찰할 수 있다.

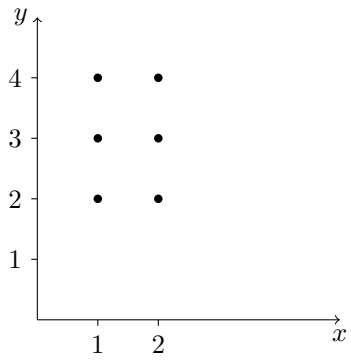


Figure 4.1: $\gamma(m^*)$의 예시

2.2 요약 뭉침 연산자

$\mathbb{M}^*$에 대해 요약 뭉침 연산자를 정의한 후 이를 이용하여 $\mathbb{S}^*$에 대한 요약 뭉침 연산자를 정의한다.

$$m_1^* \sqcup_{\mathbb{M}}^* m_2^* = \gamma(m_1^*) \cup \gamma(m_2^*)\text{에 속하는 가장 큰 } m^* \in \mathbb{M}^*$$

예를 들어, Fig. 4.2와 같이 변수가 두 개 있는 메모리에서 $m_1^* = \{x \mapsto [1,2], y \mapsto [2,4]\}$, $m_2^* = \{x \mapsto [3,5], y \mapsto [2,5]\}$ 라면, 해당 조건을 만족하는 가장 큰 직사각형은 $m_1^* \sqcup_{\mathbb{M}}^* m_2^* = \{x \mapsto [1,5], y \mapsto [2,4]\}$ 이다.

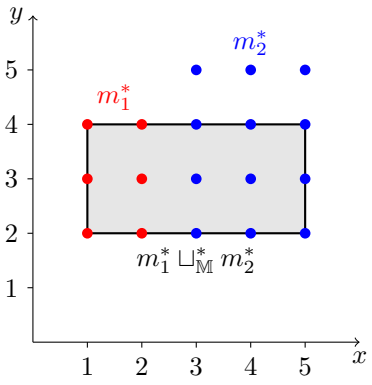


Figure 4.2: 요약 뭉침 연산자 $\sqcup_{\mathbb{M}}^*$ 예시

무한이 있는 구간에서는 크기를 비교할 수 없는데, 이 때에는 무한하지 않은 변수의 구간을 최대한 확장하도록 요약 뭉침 연산자를 정의한다.

이를 자연스럽게 라벨 별로 확장하여 $\mathbb{S}^*$에 대한 요약 뭉침 연산자를 정의한다.

2.3 요약 의미구조

위에서 정의한 next, nextTrue, nextFalse를 그대로 사용하여 $\mathbb{S}^*$ 에서의 전이관계 $\hookrightarrow^* \subseteq \mathbb{S}^* \times \mathbb{S}^*$ 를 다음과 같이 정의한다.

표현식의 요약 의미 먼저 표현식에 대한 요약 의미를 다음과 같이 정의한다.

$$\frac{}{m^* \vdash n \Downarrow [n,n]}\ \textsc{Under-E-Int} \qquad \frac{}{m^* \vdash x \Downarrow m^*(x)}\ \textsc{Under-E-Var}$$

$$\frac{m^* \vdash E_1 \Downarrow n_1^* \qquad m^* \vdash E_2 \Downarrow n_2^* \qquad n^* = n_1^* \oplus^* n_2^*}{m^* \vdash E_1 \oplus E_2 \Downarrow n^*}\ \textsc{Under-E-BinOp}$$

$$\frac{}{m^* \vdash \texttt{readInt} \Downarrow (-\infty, \infty)}\ \textsc{Under-E-ReadInt}$$

여기서 $\oplus \in \{+, -, *, <\}$ 이며, $\oplus^*$ 는 다음 조건을 만족하도록 정의된 요약 의미연산자이다.

$$\forall n \in \gamma(n_1^* \oplus^* n_2^*).\ \exists n_1 \in n_1^*, n_2 \in n_2^*.\ n = n_1 \oplus n_2$$

명령어의 요약 작은 보폭 의미구조 명령에 대한 요약 작은 보폭 의미구조는 상태 사이의 전이 관계

$$\hookrightarrow^* \subseteq\ \mathbb{S}^* \times \mathbb{S}^*$$

로 정의되며, 전이 규칙은 다음과 같다.

여기서 사용하는 filterTrue와 filterFalse는 각각 조건식 E가 참인 경우와 거짓인 경우에 해당하는 메모리 상태만 남기는 필터링 연산자이다.

$$\forall m \in \gamma(\mathsf{filterTrue}(m^*, E)).\ m \vdash E \Downarrow n \quad n \neq 0$$

$$\forall m \in \gamma(\mathsf{filterFalse}(m^*, E)).\ m \vdash E \Downarrow 0$$

좁게 어림잡는 분석에서는 위 조건을 만족하는 필터링 연산자를 사용해야 분석 틀의 조건을 만족하는 $\hookrightarrow^*$ 을 얻을 수 있다.

$$\textsc{Under-S-Skip}\ \frac{}{(\langle \ell, \texttt{skip} \rangle,\ m^*) \hookrightarrow^* (\mathsf{next}(\ell),\ m^*)} \qquad \textsc{Under-S-Assign}\ \frac{m^* \vdash E \Downarrow n^*}{(\langle \ell, x\texttt{:=}E \rangle,\ m^*) \hookrightarrow^* (\mathsf{next}(\ell),\ m^*[x \mapsto n^*])}$$

$$\textsc{Under-S-Seq}\ \frac{}{(\langle \ell, C_1; C_2 \rangle,\ m^*) \hookrightarrow^* (\mathsf{next}(\ell),\ m^*)}$$

$$\textsc{Under-S-IfTrue}\ \frac{}{(\langle \ell, \texttt{if}\ E\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \rangle,\ m^*) \hookrightarrow^* (\mathsf{nextTrue}(\ell),\ \mathsf{filterTrue}(m^*, E))}$$

$$\textsc{Under-S-IfFalse}\ \frac{}{(\langle \ell, \texttt{if}\ E\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \rangle,\ m^*) \hookrightarrow^* (\mathsf{nextFalse}(\ell),\ \mathsf{filterFalse}(m^*, E))}$$

$$\textsc{Under-S-WhileTrue}\ \frac{}{(\langle \ell, \texttt{while}\ E\ \texttt{do}\ C \rangle,\ m^*) \hookrightarrow^* (\mathsf{nextTrue}(\ell),\ \mathsf{filterTrue}(m^*, E))}$$

$$\textsc{Under-S-WhileFalse}\ \frac{}{(\langle \ell, \texttt{while}\ E\ \texttt{do}\ C \rangle,\ m^*) \hookrightarrow^* (\mathsf{nextFalse}(\ell),\ \mathsf{filterFalse}(m^*, E))}$$

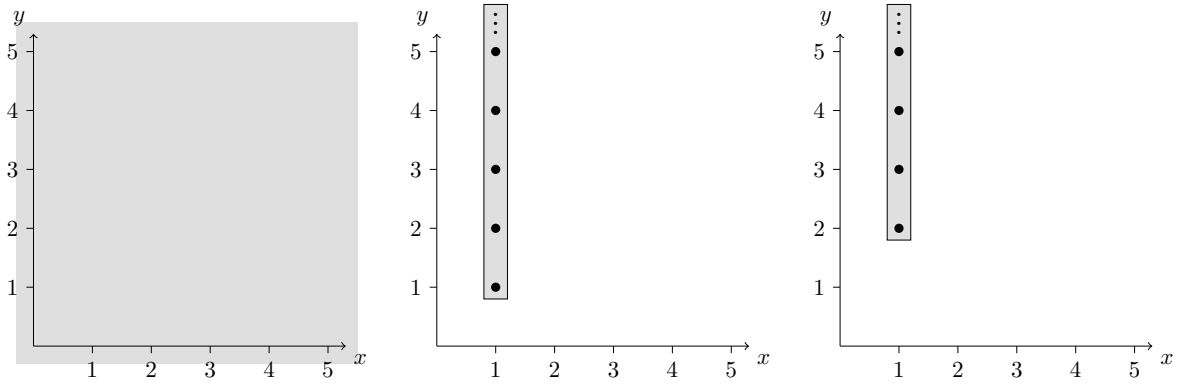
제 3 절 성공 사례

좁게 어림잡는 분석이 잘 이루어지는 예시를 살펴본다. Fig. 4.3에서 라벨은 해당하는 줄에 있는 명령어가 실행되기 전의 위치를 나타낸다. L4는 while문에서 L3의 명령어를 실행한 이후의 상태를 나타낸다. L5는 while문을 빠져나온 이후의 상태를 나타낸다.

분석은 아래와 같이 진행된다. 각 그림에서 회색 영역은 해당 위치에서의 기존의 메모리 상태, 파랑 영역은 새로 추가된 메모리 상태, 빨강 영역은 기존과 새로 추가된 메모리 상태 정보를 합친 것이다. 이와

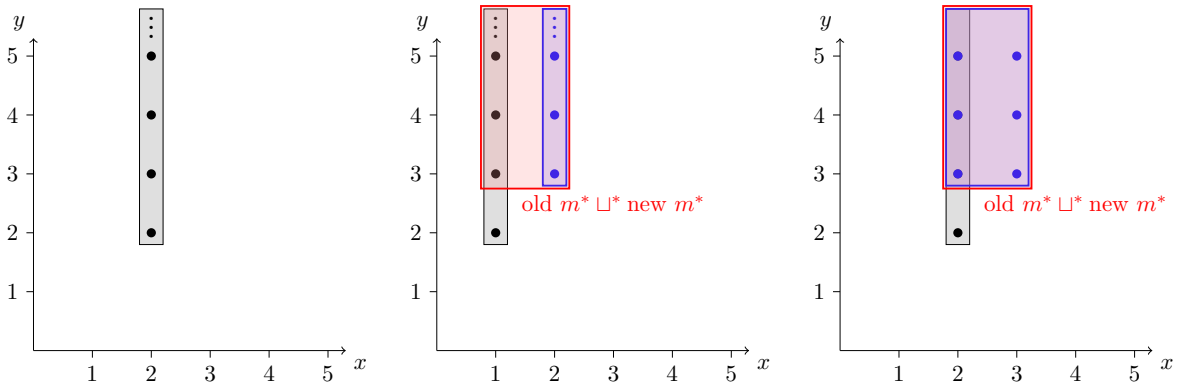
```
L0: x := 1;
L1: y := readInt;
L2: while (x < y) do
    L3: x := x + 1;
    L4:
L5:
```

Figure 4.3: 좁게 어림잡는 분석 성공 예제



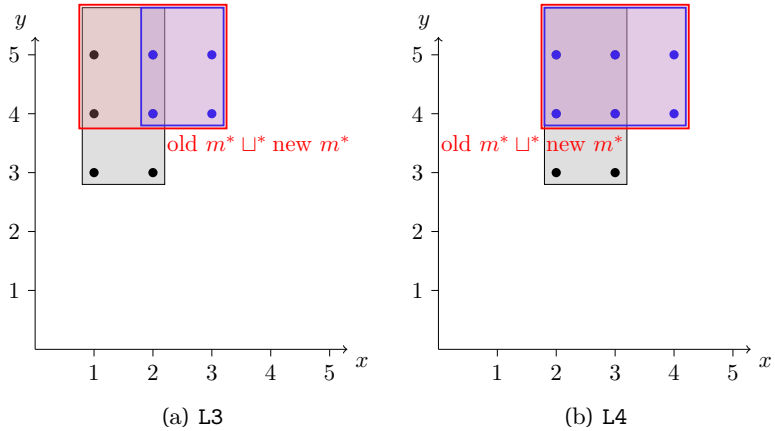
(a) L0 (b) L1, L2 (c) L3

Figure 4.4: 성공 사례, Iteration 1에서의 요약 메모리



(a) L4 (b) L3 (c) L4

Figure 4.5: 성공 사례, Iteration 2에서의 요약 메모리



(a) L3 (b) L4

Figure 4.6: 성공 사례, Iteration 3에서의 요약 메모리

같이 분석을 진행하면서, 언제든 멈춰도 그 시점까지의 결과가 실제 실행 의미에 포함됨을 보장한다.

Corollary 1에 따라, 위와 같은 분석이 진행되는 동안 언제든 멈춰도 그 시점까지의 결과가 실제 실행

의미에 포함되는 것이 보장된다. 따라서 관찰하고 싶은 성질이 관찰되었을 때, 또는 임의의 횟수만큼 반복하고 나서 언제든 분석을 멈출 수 있다.

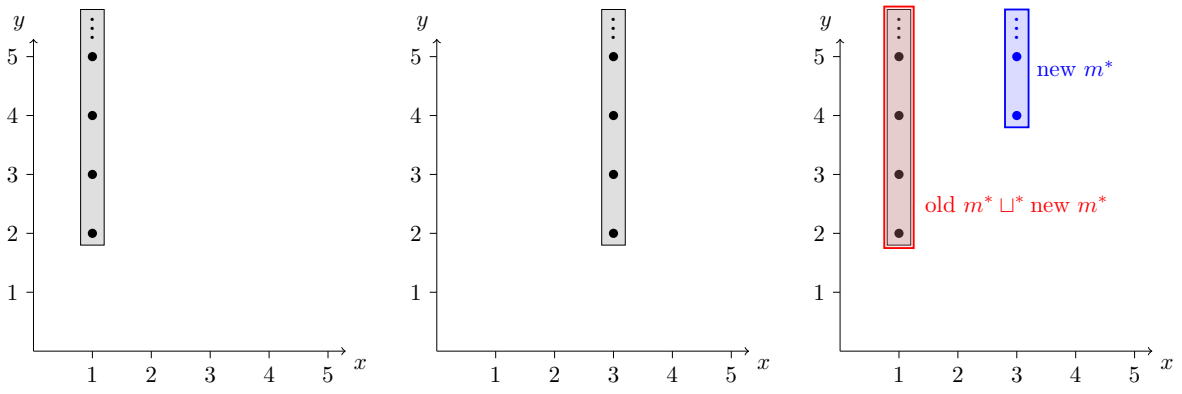
제 4 절 실패 사례

다음으로는 분석이 잘 이루어지지 않는 예시를 살펴본다. 위의 분석이 잘 이루어지는 예시 (Fig. 4.3)에서 while문 내부에서 x를 1 증가시키는 대신에 2 증가시키도록 수정한 예시를 생각해보자 (Fig. 4.7).

```
L0: x := 1;
L1: y := readInt;
L2: while (x < y) do
    L3: x := x + 2;
    L4:
L5:
```

Figure 4.7: 좁게 어림잡는 분석 실패 예제

기존과 거의 동일한 코드임에도 불구하고, 두 번째 반복에서 L3에서 과거 메모리 상태와 새로운 메모리 상태를 합쳐서 표현할 수 없게 된다. 따라서 과거 메모리를 그대로 가지고 있게 되고, 이후 반복에서도 새로운 메모리 상태가 추가되지 않으므로 분석이 더 이상 진행되지 않게 된다. 이는 Section 2.2에서 정의한 요약 뭉침 연산자가 충분히 표현력을 가지지 못해, 각 반복에서의 메모리 상태를 함께 표현하지 못하기 때문이다.



(a) L3, iteration 1 (b) L4, iteration 1 (c) L3, iteration 2

Figure 4.8: 실패 사례, Iteration 1-2에서의 요약 메모리

제 5 장 결론

본 논문에서 좁게 어림잡는 분석을 위한 요약 해석 틀을 제안하였다. 제안한 틀은 기존의 넓게 어림잡는 분석과 유사한 형태를 가진다. 먼저 기준이 되는 모둠 의미구조와 의미 공간을 정의하고, 이에 대응하는 요약 의미구조와 요약 의미 공간을 정의하였다. 이 때 제안한 틀에서 실제 실행에 포함되는 결과를 내놓는 완전한 분석이 되도록 하기 위한 조건들을 제시하였고, 계산을 중간 과정 어느 때에 멈추더라도 완전한 분석이 된다는 것을 증명하였다.

또한, 제안한 틀을 이용하여 간단한 예제에 대해 좁게 어림잡는 분석이 성공적으로 수행되는 것을 확인하였다. 그러나 조금만 바뀐 단순한 예제에서도 분석이 실패하는 문제점이 있음을 확인하였다. 이는 반복 후에 정보를 합칠 때 요약 공간의 표현력이 부족하기 때문인 것으로 보인다. 기존의 분석 틀의 구조를 그대로 유지하다 보니, 좁게 어림잡는 분석에 적합하지 않았던 것이다.

좁게 어림잡는 분석에서 의미 있는 결과를 얻기 위해서는 적합한 요약 도메인을 선택하는 것이 중요하다. Chapter 4에서는 넓게 어림잡는 분석에서 널리 사용되는 Itv 도메인을 사용했다. 그러나 실패 사례에서 살펴본 것처럼 Itv 도메인은 인접하지 않은 구간은 함께 표현할 수 없어서 실용적인 요약 뭉침 연산자를 정의하기 어렵다.

$$\mathbb{M}_1^* = Var \to \text{Itv}$$
$$\mathbb{M}_2^* = Var \to \wp(\text{Itv})$$

메모리를 요약하는 도메인을 예시에서 $\mathbb{M}_1^*$과 같이 잡은 것과 달리, 여러 개의 구간을 함께 표현할 수 있는 도메인인 $\mathbb{M}_2^*$로 잡으면 인접하지 않은 구간도 함께 표현할 수 있기 때문에 분석의 정확도가 더 향상될 수 있다.

향후 연구에서는 이러한 문제점을 분석하고 좁게 어림잡는 분석에 적합한 요약 도메인의 조건을 알아보고, 새로운 형태의 분석 틀 또는 요약 도메인을 제안하는 방향으로 연구를 진행할 예정이다.

참 고 문 헌

[1] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs," in *POPL*, 1977.

[2] P. Cousot and R. Cousot, "Systematic design of program analysis frameworks," in *POPL*, 1979.

[3] X. Rival and K. Yi, *Introduction to Static Analysis: An Abstract Interpretation Perspective*. The MIT Press, 2020.

Abstract

Static analysis is a technique for detecting potential program errors without executing the program. Traditionally, static analyzers have primarily relied on *over-approximation*, which conservatively includes all possible program executions. While such analyses are effective in proving the absence of errors, they suffer from the limitation that reported errors are not guaranteed to occur in actual executions. As a result, analysis results often contain many false alarms, which reduces their practical usefulness.

To address this limitation, this thesis focuses on *under-approximation*–based static analysis, which includes only a subset of actual executions while guaranteeing that every execution reported by the analysis can indeed occur at runtime. In particular, we propose a general analysis framework for systematically designing under-approximate analyzers, based on the semantic-function–oriented structure commonly used in the abstract interpretation framework.

We define the program semantics using small-step semantics and collecting semantics, review the conventional over-approximate analysis framework built upon them, and introduce an under-approximate abstract semantics that reverses the direction of the inclusion relation. We identify sufficient conditions on the initial states, abstract semantic function, and abstract join operator, under which analysis results are contained in the concrete program semantics and remain semantically valid even if the analysis is terminated early.

Furthermore, we apply the proposed framework to a simple imperative language and design an under-approximate analyzer. Through illustrative examples, we examine the characteristics and limitations of under-approximate analysis in practice, and show that the expressiveness of the abstract domain plays a crucial role in the precision and progress of under-approximate analysis. In particular, we discuss the limitations of the widely used *interval* domain in the context of under-approximation.

By formalizing under-approximate static analysis within a semantic-function–based framework, this thesis provides a theoretical foundation for analysis results that witness the actual existence of errors, and establishes a basis for the complementary use of over-approximate and under-approximate analyses.

Keywords: static analysis, program analysis, abstract interpretation, under-approximation, completeness