이제 이와 같은 요약해석의 원리/틀에 따라 프로그램 분석을 구체적으로 디자인해보자. 그 틀에 들어가는 프로그램 분석은 다양한 스타일이 될 수 있다; 분석 할 프로그램이 짜여진 언어에 따라서, 분석할 성질에 따라서, 그리고 프로그램 실행의 의미를 표현하는 방식에 따라서.

진행전에 우선 몇가지 유용한 사실들을 살펴보자.

6.1.1.1 요약에 관한 사실들

"$\hat{D}$는 D를 요약한 것이다"는 것은 둘 사이를 갈로아 연결

$$D \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{D}$$

시키는 요약함수 $\alpha : D \to \hat{D}$ 와 구체함수 $\gamma : \hat{D} \to D$ 가 있을 때를 말한다.

Example 24 $(2^{\mathbb{Z}}, \subseteq, \emptyset)$를 $(\{\bot, +, -, 0, \top\}, \sqsubseteq, \bot)$로 요약할 수 있다. □

Example 25 $(2^{\mathbb{Z}}, \subseteq, \emptyset)$를 요약한 공간은 $(\{\bot, +, -, 0\}, \sqsubseteq, \bot)$이 될 수 없다. 갈로아 연결이 불가능하다. 반드시 $\mathbb{Z}$를 의미하는 요약원소가 있어야 한다. □

Fact 6 *갈로아 연결은 조립식이다. A를 요약한 것이 $\hat{A}$이고 B를 요약한 것이 $\hat{B}$이면,*

- *$A \times B$를 $\hat{A} \times \hat{B}$로 요약할 수 있다.*
- *$A + B$를 $\hat{A} + \hat{B}$로 요약할 수 있다.*
- *$A \to B$를 $\hat{A} \to \hat{B}$로 요약할 수 있다.*

증명은 쉽다.

Fact 7 *D와 $\hat{D}$를 갈로아 연결시키는 요약함수와 구체함수는 대게가 하나가 정의되면 다른 하나는 저절로 정의된다.*

- $\alpha : D \to \hat{D}$가 연속(continuous) 함수이고 D가 임의의 부분 집합의 최소 윗뚜껑(least upper bound)이 있으면($\sqcup$-complete), 갈로아 연결 짝 γ는 α로 부터 다음과 같이 정의된다:

$$\gamma\,\hat{x} = \sqcup\{x \mid \alpha\,x \sqsubseteq \hat{x}\}.$$

- $\gamma : \hat{D} \to D$가 연속(continuous) 함수이고 $\hat{D}$가 임의의 부분 집합의 최대 밑뚜껑(greatest lower bound)이 있으면($\sqcap$-complete), 갈로아 연결 짝 α는 γ로 부터 다음과 같이 정의된다:

$$\alpha\,x = \sqcap\{\hat{x} \mid x \sqsubseteq \gamma\,\hat{x}\}.$$

증명은 쉽다.

Fact 8 안전한 함수들로 조립한 함수는 안전하다. 즉, $A \underset{\alpha_A}{\overset{\gamma_A}{\leftrightarrows}} \hat{A}$, $B \underset{\alpha_B}{\overset{\gamma_B}{\leftrightarrows}} \hat{B}$, $C \underset{\alpha_C}{\overset{\gamma_C}{\leftrightarrows}} \hat{C}$ 이고, 단조(monotonic) 함수 $f : A \to B$, $\hat{f} : \hat{A} \to \hat{B}$, $g : B \to C$, $\hat{g} : \hat{B} \to \hat{C}$가

$$\alpha_B \circ f \sqsubseteq \hat{f} \circ \alpha_A \quad \text{이고} \quad \alpha_C \circ g \sqsubseteq \hat{g} \circ \alpha_B$$

이면

$$\alpha_C \circ (g \circ f) \sqsubseteq (\hat{g} \circ \hat{f}) \circ \alpha_A$$

이다. 증명은 쉽다.

Fact 9 조립식으로 함수를 요약하는 경우, 요약한 함수들로 조립한 함수는 안전하다. 즉, 모든 함수 요약 $X \to Y \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{X} \to \hat{Y}$들이 X와 Y의 갈로아 연결들을 가지고

$$\alpha(f) = \alpha_Y \circ f \circ \gamma_X$$

로 정의된다고 하자. $A \underset{\alpha_A}{\overset{\gamma_A}{\leftrightarrows}} \hat{A}$, $B \underset{\alpha_B}{\overset{\gamma_B}{\leftrightarrows}} \hat{B}$, $C \underset{\alpha_C}{\overset{\gamma_C}{\leftrightarrows}} \hat{C}$ *이고, 단조(*monotonic*) 함수* $f : A \to B$ *와* $g : B \to C$ *에 대해서,*

$$\alpha(g \circ f) \sqsubseteq \alpha(g) \circ \alpha(f)$$

이다. 증명은 쉽다.

6.1.2 요약해석 적용$_0$

다음의 정수식 언어로 짜여진 프로그램의 분석기를 디자인하자. 너무 간단해서 요약해석 틀까지 필요없을 것이지만, 요약해석 틀을 이용하는 첫 시연으로 적절하다.

$$\begin{array}{rcll} E & \to & n & (n \in \mathbb{Z}) \\ & | & E \texttt{ + } E & \\ & | & \texttt{-} E & \\ & | & \texttt{if } E \texttt{ then } E \texttt{ else } E & \end{array}$$

분석할 성질은 계산되는 정수값의 부호라고 하자.

귀납적인 분석 룰로 쓰면 다음과 같을 것이고, 그 룰이 옳다는 것을 보일 수 있고, 그 룰을 구현하는 것도 곧바로 룰 따라 재귀적으로 호출하면 될 것이다:

$$\frac{}{n : sign(n)} \qquad \frac{E_1 : s_1 \quad E_2 : s_2}{E_1 \texttt{ + } E_2 : s_1 \oplus s_2} \qquad \frac{E : s}{\texttt{-} E : \ominus s} \qquad \frac{E_2 : s_2 \quad E_3 : s_3}{\texttt{if } E_1 \texttt{ then } E_2 \texttt{ else } E_3 : s_2 \sqcup s_3}$$

여기서

$$\begin{array}{rcl} sign(n \geq 0) & = & + \\ sign(n < 0) & = & - \end{array}$$

$$\begin{array}{rcl} \ominus(+) & = & - \\ \ominus(-) & = & + \end{array}$$

이고,

$\oplus$	$+$	$-$
$+$	$+$	$\top$
$-$	$\top$	$-$

올바른 이유는 $+, -, \top$이 의미하는 바가 각각

$$\begin{aligned}\gamma(+) &= \{n \in \mathbb{Z} \mid n \geq 0\} \\ \gamma(-) &= \{n \in \mathbb{Z} \mid n < 0\} \\ \gamma(\top) &= \mathbb{Z}\end{aligned}$$

이고, 모든 정수식 E에 대해서

$$E : s \Rightarrow val(E) \in \gamma(s)$$

이기 때문이다. 이것을 쉽게 증명할 수 있을 것이다. E에 대한 구조적 귀납법으로.

이것을 요약해석 틀에 끼어 넣어보자. 정수식의 의미함수(위에서 val)를 정의하면

$$\mathcal{V} : Exp \to 2^{\mathbb{Z}}$$

이고

$$\begin{aligned}\mathcal{V}\, n &= \{n\} \\ \mathcal{V}\, E_1 \texttt{ + } E_2 &= \{z_1 + z_2 \mid z_i \in \mathcal{V}\, E_i\} \\ \mathcal{V}\texttt{ - }E &= \{-z \mid z \in \mathcal{V}\, E\} \\ \mathcal{V}\ \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 &= 0 \in \mathcal{V}\, E_1\, ?\, \mathcal{V}\, E_2\ :\ \mathcal{V}\, E_3\end{aligned}$$

두가지 의문이 있을 수 있다:

- 식 E의 의미는 $\mathcal{V}\,E$로 정의되는데, 이것이 왜 어느 연속함수의 최소고정점인가?

  $\mathcal{V}\,E$는 E의 구조에 따라 귀납적으로 정의된다. 이것은 바로 상수함수

  $$\lambda x.\mathcal{V}\,E \quad \in \quad 2^{\mathbb{Z}} \to 2^{\mathbb{Z}}$$

  의 유일한 고정점(*fixpoint*)이다. 위의 함수는 상수함수이므로 연속함수이다.

- 왜 실제 의미 영역이 $\mathbb{Z}$가 아니고 $2^{\mathbb{Z}}$인가? 분석할 때 사용하는 요약된 세계와 갈로아 연결을 해보려니 그렇다. $2^{\mathbb{Z}}$는 $\mathbb{Z}$의 부분집합들의 집합이고, 최소 원소는 $\emptyset$, 순서는 집합의 포함순서가 되겠다. CPO가 된다.

요약된 세계 $\hat{\mathbb{Z}}$은 $\{+, -, \bot, \top\}$ 이고, 원소들 사이의 순서는 $\bot \sqsubseteq + \sqsubseteq \top$이고 $\bot \sqsubseteq - \sqsubseteq \top$이다. 유한한 집합이고 최소원소가 있다. CPO다.

갈로아 연결은 다음과 같다:

$$\begin{aligned} \alpha\{\} &= \bot \\ \alpha\{n \mid n \geq 0\} &= + \\ \alpha\{n \mid n < 0\} &= - \\ \alpha Z &= \top \end{aligned}$$

이고

$$\begin{aligned} \gamma + &= \{n \in \mathbb{Z} \mid n \geq 0\} \\ \gamma - &= \{n \in \mathbb{Z} \mid n < 0\} \\ \gamma \top &= \mathbb{Z}. \end{aligned}$$

요약된 세계에서의 의미함수는 다음과 같이 정의된다:

$$\begin{aligned}
\hat{\mathcal{V}}\, n &= \alpha\,\{n\} \\
\hat{\mathcal{V}}\, E_1 \texttt{ + } E_2 &= (\hat{\mathcal{V}}\, E_1) \oplus (\hat{\mathcal{V}}\, E_2) \\
\hat{\mathcal{V}}\, \texttt{-}\, E &= \ominus(\hat{\mathcal{V}}\, E) \\
\hat{\mathcal{V}}\, \texttt{if } E_1 \texttt{ then } E_2 \texttt{ else } E_3 &= \hat{\mathcal{V}}\, E_2 \sqcup \hat{\mathcal{V}}\, E_3
\end{aligned}$$

맞는지 확인할 것은, 모든 정수식 E에 대해서:

$$\alpha \circ (\lambda x.\mathcal{V}\, E)) \sqsubseteq (\lambda \hat{x}.\hat{\mathcal{V}}\, E) \circ \alpha$$

단순히 하면,

$$\alpha(\mathcal{V}\, E)) \sqsubseteq \hat{\mathcal{V}}\, E.$$

증명은 E에 대한 구조적인 귀납법으로 진행되고, α의 정의를 이용해서 경우를 따지면 된다.

분석의 구현은 그냥 $\hat{\mathcal{V}}\, E$를 계산하면 되겠다. 항상 끝난다.

6.1.3 요약해석 적용$_1$

위의 정수식 언어를 확장하자. 변수를 사용할 수 있는 정수식이다.

E	$\rightarrow$	n	$(n \in \mathbb{Z})$
	$\mid$	E + E	
	$\mid$	- E	
	$\mid$	x	변수
	$\mid$	let x = E_1 in E_2	지역 변수

let에 의해 정의되지 않고 쓰이는 변수는 입력변수라고 하자. 입력은 정수만 가능하다고하고.

환경

$$\sigma \in Env = Var \xrightarrow{\text{fin}} 2^{\mathbb{Z}}$$

이고, 의미함수

$$\mathcal{V} : Exp \to Env \to 2^{Z}$$

는 아래와 같이 정의될 것이다:

$$\begin{aligned}
\mathcal{V}\, n\, \sigma &= \{n\} \\
\mathcal{V}\, x\, \sigma &= \sigma\, x \\
\mathcal{V}\, E_1 \texttt{ + } E_2\, \sigma &= \{z_1 + z_2 \mid z_i \in \mathcal{V}\, E_i\, \sigma\} \\
\mathcal{V}\, \texttt{-}\, E\, \sigma &= \{-z \mid z \in \mathcal{V}\, E\, \sigma\} \\
\mathcal{V}\, \texttt{let } x \texttt{ = } E_1 \texttt{ in } E_2\, \sigma &= \mathcal{V}\, E_2\, \sigma\{x \mapsto \mathcal{V}\, E_1\, \sigma\}
\end{aligned}$$

주어진 식 E의 의미는 E의 구조에 따라 조립식으로 구성된 $\mathcal{V}\,E$로 (상수 함수 $\lambda x.\mathcal{V}\,E$의 최소 고정점으로) 정의된다.

요약된 의미함수를

$$\hat{\mathcal{V}} : Exp \to \hat{Env} \to \hat{\mathbb{Z}}$$

정의하고, 갈로아 연결

$$Env \to 2^{Z} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{Env} \to \hat{\mathbb{Z}}$$

을 조립식으로 구성하자. $Env \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \hat{Env}$사이의 갈로아 연결과 $2^{\mathbb{Z}} \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \hat{\mathbb{Z}}$사이의 갈로아 연결을 가지고:

$$\begin{aligned}
\alpha\, f &= \alpha_2 \circ f \circ \gamma_1 \\
\gamma\, \hat{f} &= \gamma_2 \circ \hat{f} \circ \alpha_1
\end{aligned}$$

요약된 세계에서의 의미함수는 다음과 같이 정의된다:

$$\begin{aligned}
\hat{\mathcal{V}}\, n\, \hat{\sigma} &= \alpha_2\, \{n\} \\
\hat{\mathcal{V}}\, E_1 \texttt{ + } E_2\, \hat{\sigma} &= (\hat{\mathcal{V}}\, E_1\, \hat{\sigma}) \oplus (\hat{\mathcal{V}}\, E_2\, \hat{\sigma}) \\
\hat{\mathcal{V}}\, \texttt{-}\, E\, \hat{\sigma} &= \ominus(\hat{\mathcal{V}}\, E\, \hat{\sigma}) \\
\hat{\mathcal{V}}\, \texttt{let } x \texttt{ = } E_1 \texttt{ in } E_2\, \hat{\sigma} &= \hat{\mathcal{V}}\, E_2\, \hat{\sigma}\{x \mapsto \hat{\mathcal{V}}\, E_1\, \hat{\sigma}\}
\end{aligned}$$

맞는지 확인할 것은, 모든 정수식 E에 대해서:

$$\alpha \circ (\lambda x.\mathcal{V}\, E)) \sqsubseteq (\lambda \hat{x}.\hat{\mathcal{V}}\, E) \circ \alpha$$

단순히 하면,

$$\alpha(\mathcal{V}\, E) \sqsubseteq \hat{\mathcal{V}}\, E. \tag{6.13}$$

증명은 E에 대한 구조적인 귀납법으로 진행되고, α의 정의를 이용해서 경우를 따지면 된다. 여기까지가 요약해석의 틀이다.

이제 문제는 분석의 구현이다. 즉, 함수 $\hat{\mathcal{V}}\, E \in \hat{Env} \to \hat{\mathbb{Z}}$를 계산해 내야 한다. 이 함수를 계산하는 방법은 두가지가 있겠다.

- 만일 $\hat{Env}$의 원소가 유한하다면, 모든 $\hat{\sigma} \in \hat{Env}$에 대해서 $\hat{\mathcal{V}}\, E\, \hat{\sigma}$를 각각 계산해서, 테이블을 만들면 된다. $\hat{\mathcal{V}}\, E\, \hat{\sigma}$의 계산은 항상 끝난다($E$의 구조에 따라 조립식으로 정의되므로).

  정수식 E의 의미가 함수인 것은 적절하다. 정수식에 자유변수가 있다면 그 변수의 값을 가지고 있는 환경에 따라서 계산되는 값이 다를 수 있으므로.

  이렇게 모든 경우를 계산할 필요가 없을 수 있다. 분석 시간이 너무 많이 들 수도 있고, 아예 불가능 할 수 도 있기 때문이다.

- 만일 $\hat{Env}$의 원소가 무한히 많다면? 혹은 유한하지만 모든 경우를 계산할 필요가 없을 경우는?

하나의 요약 환경으로 프로그램 식의 자유 변수가 가질 수 있는 값(초기 환경)을 모두 포섭하게 할 수 있다. 자유 변수들이 가지는 값을 $\top$으로 가지고 있는 환경이 되겠다. 그러한 환경 $\hat{\sigma}_0$에 대해서 $\hat{\mathcal{V}}\,E\hat{\sigma}_0$을 계산하면 된다.

이것이 왜 올바른 구현인가?

Proof. 안전성(식 (6.13))이

$$\alpha(\mathcal{V}\,E) \sqsubseteq \hat{\mathcal{V}}\,E$$

증명되었으므로,

$$(\alpha(\mathcal{V}\,E))\hat{\sigma}_0 \sqsubseteq (\hat{\mathcal{V}}\,E)\hat{\sigma}_0$$

다시, 요약함수 α의 정의해 의해서

$$\alpha_2((\mathcal{V}\,E)\,(\gamma_1\,\hat{\sigma}_0)) \sqsubseteq (\hat{\mathcal{V}}\,E)\hat{\sigma}_0$$

이 성립한다. 이는 갈로아 연결로 인해,

$$(\mathcal{V}\,E)\,(\gamma_1\,\hat{\sigma}_0) \sqsubseteq \gamma_2((\hat{\mathcal{V}}\,E)\hat{\sigma}_0)$$

즉, 분석결과 $(\hat{\mathcal{V}}\,E)\hat{\sigma}_0$는 실제 상황 $\mathcal{V}\,E\,(\gamma_1\,\hat{\sigma}_0)$을 포섭한다는 결론이 나온다. 따라서 위와 같은 구현은 옳다. □

반드시 하나의 요약환경으로 모든 초기 환경을 커버하게 할 필요도 없다. 몇개의 요약환경 $\{\hat{\sigma}_1, \cdots, \hat{\sigma}_k\}$으로 모든 초기 환경의 경우가 커버된다면 각각의 요약환경에 대해서 $\hat{\mathcal{V}}\,E\sigma_i$를 계산해도 될 것이다. 위에서 논의한 대로, 각각의 경우는 각 요약환경이 의미하는 실세계에서의 프로그램식의 의미를 포섭하므로, 모든 요약환경이 실세계의 모두를 커버하는 경우라면, 분석된 프로그램식의 의미들은 뭉치로 실세계를 모두 커버하게 된다.

6.1.4 요약해석 적용$_2$

이제는 함수 정의와 호출이 있는 언어를 생각하자. 함수는 재귀호출될 수 있고, 함수가 자유롭게 흘러다닐 수 있는(*higher-order*) 언어가 되겠다.

$$
\begin{array}{rcll}
E & \rightarrow & n & (n \in \mathbb{Z}) \\
 & | & E + E & \\
 & | & - E & \\
 & | & x & \text{변수} \\
 & | & f\lambda x.E & \text{재귀 함수} \\
 & | & E\ E & \text{함수 호출}
\end{array}
$$

임의의 프로그램식의 의미가 어떻게 함수의 고정점으로 정의되는지 보자. 두 가지 방안이 있을 수 있다.

- 궁극의 스타일(*denotational semantics*)로 의미를 정의하는 것이다. 프로그램의 의미는 조립식으로 정의된다. 이 방법에서 프로시져의 의미는 함수로 정의된다. 그 프로시져가 재귀적인 것이라면, 그 프로시져의 의미는 어느 함수의 최소고정점으로 정의된다. 이 최소고정점 들이 프로그램의 의미 정의속 여기저기 나타난다.

  그 의미의 요약본도 비슷하게, 조립식으로 정의될 것이고 그 안에 재귀 프로시져의 의미를 정의하는 최소고정점이 여기저기 나타난다. 이 요약본이 원래의 것을 안전하게 요약한 것이라고 증명하는 데는 고정점 귀납법(*fixpoint induction*)이 필요하게 될 것이다. 고정점을 품은 대상들에 대한 성질을 증명하는 것이므로.

- 다른 방식의 의미 정의는, 과정을 드러내는 방식(*operational semantics*)을 함수 꼴로 표현하는 것이다. 이렇게 되면, 프로그램의 의미가 조립식으로 정의되지 않지만 정의안에 최소고정점이 나타나지는 않는다.

  이때 의미함수는 프로그램 실행기(*interpreter*) 함수를 재귀적으로 표현한 모습이 된다. 이렇게 되면, 의미함수는 프로그램의 의미를 결정하는 함수

가 아니라, 의미에 대한 방정식을 결정하는 함수가 된다.

두번째 방식으로 가보자.

환경은

$$\begin{aligned} \sigma \in Env &= Var \xrightarrow{\text{fin}} Val \\ Val &= 2^{\mathbb{Z}} + 2^{Closure} \\ Closure &= Exp \times Env \end{aligned}$$

이고, 의미함수

$$\mathcal{V} : Exp \to Env \to Val$$

는 아래와 같이 재귀적으로 표현될 것이다:

$$\begin{aligned} \mathcal{V}\, n\, \sigma &= \{n\} \\ \mathcal{V}\, x\, \sigma &= \sigma\, x \\ \mathcal{V}\, E_1 \texttt{+} E_2\, \sigma &= \{z_1 + z_2 \mid z_i \in \mathcal{V}\, E_i\, \sigma\} \\ \mathcal{V}\, \texttt{-}\, E\, \sigma &= \{-z \mid z \in \mathcal{V}\, E\, \sigma\} \\ \mathcal{V}\, f\lambda x.E\, \sigma &= \{\langle f\lambda x.E, \sigma\rangle\} \\ \mathcal{V}\, E_1\, E_2\, \sigma &= \cup\{\mathcal{V}\, E\, \sigma'\{x \mapsto (\sigma'\, x) \cup (\mathcal{V}\, E_2\, \sigma)\}\{f \mapsto \langle f\lambda x.E, \sigma'\rangle\} \mid \langle f\lambda x.E, \sigma'\rangle \in \mathcal{V}\, E_1\, \sigma\} \end{aligned}$$

위의 것은 프로그램 E의 의미 $\mathcal{V}\, E$를 정의하는가? 그렇지 않다. 프로그램 E의 의미 $\mathcal{V}\, E$는 조립식으로 정의되지 않기 때문이다. E가 재귀함수를 호출하고 그 함수내에 재귀호출 "$f\ E'$"이 있는 경우, 정의하지 못하고 만다:

$$\begin{aligned} \mathcal{V}\, E &= \cdots \mathcal{V}\, f\ E' \cdots \\ &= \cdots (\cdots \mathcal{V}\, f\ E' \cdots) \cdots \\ &= \cdots \end{aligned}$$

따라서, 위에서 표현한 $\mathcal{V}$은 E를 가지고 만드는 방정식을 표현하는 것 뿐이고, 그 방정식의 해가 프로그램의 E의 의미 $\mathcal{V}\, E$가 되는 것이다. 식 E의 방정식

을 만들어주는 함수

$$\mathcal{F} : (Exp \to Env \to Val) \to (Exp \to Env \to Val)$$

는 다름아니라:

$$\begin{aligned}
\mathcal{F}\,\mathcal{V}\,n\,\sigma &= \{n\} \\
\mathcal{F}\,\mathcal{V}\,x\,\sigma &= \sigma\,x \\
\mathcal{F}\,\mathcal{V}\,E_1 \texttt{ + } E_2\,\sigma &= \{z_1 + z_2 \mid z_i \in \mathcal{V}\,E_i\,\sigma\} \\
\mathcal{F}\,\mathcal{V}\,\texttt{-}\,E\,\sigma &= \{-z \mid z \in \mathcal{V}\,E\,\sigma\} \\
\mathcal{F}\,\mathcal{V}\,f\lambda x.E\,\sigma &= \{\langle f\lambda x.E, \sigma\rangle\} \\
\mathcal{F}\,\mathcal{V}\,E_1\,E_2\,\sigma &= \cup\{\mathcal{V}\,E\,\sigma'\{x \mapsto \mathcal{V}\,E_2\,\sigma\}\{f \mapsto \langle f\lambda x.E, \sigma'\rangle\} \mid \langle f\lambda x.E, \sigma'\rangle \in \mathcal{V}\,E_1\,\sigma\}
\end{aligned}$$

이 된다.

정리하면, 프로그램 E의 의미는 그 의미 $\mathcal{V}\,E$에 대한 방정식

$$\mathcal{V}\,E = \mathcal{F}\,\mathcal{V}\,E$$

과 E안의 모든 식 E_i들의 의미 $\mathcal{V}\,E_i$에 대한 방정식

$$\mathcal{V}\,E_i = \mathcal{F}\,\mathcal{V}\,E_i$$

들의 해를 가지고 정의된다.

그러한 $\mathcal{F}$를 통해서 프로그램 E로 부터 도출된 연립방정식을

$$Eqn(E, \mathcal{F})$$

라고 쓰자. 그 연립방정식을 하나의 함수 $\mathcal{F}_E$를 이용해서 표현하면

$$\begin{pmatrix} X_0 \\ \vdots \\ X_n \end{pmatrix} = \mathcal{F}_E \begin{pmatrix} X_0 \\ \vdots \\ X_n \end{pmatrix}$$

이 되고, 방정식의 주인공 X_i는 $\mathcal{V}\, E_i$대신에 쓴 것이고, 프로그램 E의 모든 하부 식들은 n개가 있다고 한 것이다. ($\mathcal{F}_E$의 정의는 $\mathcal{F}$로 부터 명백하다.)

Example 26 다음 프로그램

$$1 + (k\lambda x.k\ (-\ x))\ 0$$

을 생각하자. 위 프로그램의 모든 부품식들마다 번호를 붙이자.

$$\underbrace{1}_{1} + (k\lambda x.\ \underbrace{k}_{6}\ \underbrace{(-\ \underbrace{x}_{8})}_{7}\ \underbrace{0}_{4})$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{0}$$

각 프로그램 식 E_i에 대해서 방정식

$$\mathcal{V}\, E_i = \mathcal{F}\, \mathcal{V}\, E_i$$

을 모은 연립방정식은

$$
\begin{aligned}
\mathcal{V}\,E_0 &= \lambda\sigma.\{z_1 + z_2 \mid z_1 \in \mathcal{V}\,E_1\,\sigma, z_2 \in \mathcal{V}\,E_2\,\sigma\} \\
\mathcal{V}\,E_1 &= \lambda\sigma.\{1\} \\
\mathcal{V}\,E_2 &= \lambda\sigma.\cup\{\mathcal{V}\,E\,\sigma'\{x \mapsto \mathcal{V}\,E_4\,\sigma\}\{f \mapsto \langle f\lambda x.E, \sigma'\rangle\} \mid \langle f\lambda x.E, \sigma'\rangle \in \mathcal{V}\,E_3\,\sigma\} \\
\mathcal{V}\,E_3 &= \lambda\sigma.\{\langle k\lambda x.E_5, \sigma\rangle\} \\
\mathcal{V}\,E_4 &= \lambda\sigma.\{0\} \\
\mathcal{V}\,E_5 &= \lambda\sigma.\cup\{\mathcal{V}\,E\,\sigma'\{x \mapsto \mathcal{V}\,E_7\,\sigma\}\{f \mapsto \langle f\lambda x.E, \sigma'\rangle\} \mid \langle f\lambda x.E, \sigma'\rangle \in \mathcal{V}\,E_6\,\sigma\} \\
\mathcal{V}\,E_6 &= \lambda\sigma.\sigma\,k \\
\mathcal{V}\,E_7 &= \lambda\sigma.\{-z \mid z \in \mathcal{V}\,E_8\,\sigma\} \\
\mathcal{V}\,E_8 &= \lambda\sigma.\sigma\,x
\end{aligned}
$$

방정식의 주인공을 $\mathcal{V}\,E_i$ 대신에 X_i로 다시쓰면:

$$
\begin{aligned}
X_0 &= \lambda\sigma.\{z_1 + z_2 \mid z_1 \in X_1\,\sigma, z_2 \in X_2\,\sigma\} \\
X_1 &= \lambda\sigma.\{1\} \\
X_2 &= \lambda\sigma.\cup\{X_E\,\sigma'\{x \mapsto X_4\,\sigma\}\{f \mapsto \langle f\lambda x.E, \sigma'\rangle\} \mid \langle f\lambda x.E, \sigma'\rangle \in X_3\,\sigma\} \\
X_3 &= \lambda\sigma.\{\langle k\lambda x.X_5, \sigma\rangle\} \\
X_4 &= \lambda\sigma.\{0\} \\
X_5 &= \lambda\sigma.\cup\{X_E\,\sigma'\{x \mapsto X_7\,\sigma\}\{f \mapsto \langle f\lambda x.E, \sigma'\rangle\} \mid \langle f\lambda x.E, \sigma'\rangle \in X_6\,\sigma\} \\
X_6 &= \lambda\sigma.\sigma\,k \\
X_7 &= \lambda\sigma.\{-z \mid z \in X_8\,\sigma\} \\
X_8 &= \lambda\sigma.\sigma\,x
\end{aligned}
$$

위의 방정식이 특이한 것은 방정식의 주인공들이 완전히 드러나 있지 않다는 것이다. X_2의 방정식을 보면, "X_E"가 나타나는 데 프로그램 식 E의 정체는

X_3의 해가 밝혀질 때

$$\text{“}\langle f\lambda x.E, \sigma'\rangle \in X_3\,\sigma''$$

비로소 알 수 있다. 방정식 자체가 방정식을 푸는 중에 결정되는 셈인데, 이것은 함수가 프로그램의 값이 되어 자유롭게 흘러다닐 수 있는(*higher-order*) 언어를 대상으로 분석하는 것이기 때문에 생기는 현상이다. 어떤 함수가 불릴 수 있는 지는 분석해 봐야 알 수 있는 것이고, 분석하려면 어떤 함수가 불릴 수 있는 지를 알아야 하기 때문이다. □

연립 방정식 $Eqn(E,\mathcal{F})$의 최소의 해는 항상 존재한다. 왜냐하면, 그러한 해는 연속 함수 $\mathcal{F}$의 최소 고정점

$$lfp\mathcal{F} \in Exp \to Env \to Val$$

의 엔트리들과 일치

$$\forall E_i \in E : lfp\mathcal{F}\ E_i = \text{연립 방정식 } Eqn(E,\mathcal{F})\text{에서 } \mathcal{V}\,E_i\text{의 해} \tag{6.14}$$

하기 때문이다.

연립방정식 $Eqn(E,\mathcal{F})$의 해를 $[\![Eqn(E,\mathcal{F})]\!]$ 로 쓰고, 해답 중에서 $\mathcal{V}\,E_i$의 답을 $[\![Eqn(E,\mathcal{F})]\!]\,E_i$로 쓰자. 이 방식으로 (6.14)를 다시 쓰면

$$\forall E_i \in E : lfp\mathcal{F}\ E_i = [\![Eqn(E,\mathcal{F})]\!]\,E_i.$$

프로그램 E의 요약된 의미는 다음의 요약공간

$$\begin{aligned} \hat{\sigma} \in \hat{Env} &= Var \xrightarrow{\text{fin}} \hat{Val} \\ \hat{Val} &= \hat{\mathbb{Z}} + \hat{Closure} \\ \hat{Closure} &= 2^{Exp} \end{aligned}$$

을 가지고 정의된 $\hat{\mathcal{F}}$ 가 만들어 내는 연립방정식 $Eqn(E, \hat{\mathcal{F}})$의 해가 된다. $\hat{\mathcal{F}}$는

$$\hat{\mathcal{F}} \in (Exp \to \hat{Env} \to \hat{Val}) \to (Exp \to \hat{Env} \to \hat{Val})$$

이고 그 정의는

$$\begin{aligned}
\hat{\mathcal{F}}\,\hat{\mathcal{V}}\,n\,\hat{\sigma} &= \alpha_2\,\{n\} \\
\hat{\mathcal{F}}\,\hat{\mathcal{V}}\,x\,\hat{\sigma} &= \hat{\sigma}\,x \\
\hat{\mathcal{F}}\,\hat{\mathcal{V}}\,E_1 + E_2\,\hat{\sigma} &= (\hat{\mathcal{V}}\,E_1\,\hat{\sigma}) \oplus (\hat{\mathcal{V}}\,E_2\,\hat{\sigma}) \\
\hat{\mathcal{F}}\,\hat{\mathcal{V}} - E\,\hat{\sigma} &= \ominus(\hat{\mathcal{V}}\,E\,\hat{\sigma}) \\
\hat{\mathcal{F}}\,\hat{\mathcal{V}}\,\lambda x.E\,\hat{\sigma} &= \{\lambda x.E\} \\
\hat{\mathcal{F}}\,\hat{\mathcal{V}}\,E_1\,E_2\,\hat{\sigma} &= \sqcup\{\hat{\mathcal{V}}\,E\,\hat{\sigma}\{x \mapsto \hat{\sigma}\,x \cup \hat{\mathcal{V}}\,E_2\,\hat{\sigma}\} \mid \lambda x.E \in \hat{\mathcal{V}}\,E_1\,\hat{\sigma}\}
\end{aligned}$$

여기서 $\hat{\mathcal{F}}$는 연속 함수이므로, $lfp\hat{\mathcal{F}}$는 존재하고, 연립 방정식 $Eqn(E, \hat{\mathcal{F}})$의 해는 항상

$$lfp\hat{\mathcal{F}} \in Exp \to \hat{Env} \to \hat{Val}$$

의 엔트리들과 일치한다:

$$\forall E_i \in E : lfp\hat{\mathcal{F}}\,E_i = [\![Eqn(E, \hat{\mathcal{F}})]\!]\,E_i.$$

정리하면, 프로그램 E의 실제 의미 $[\![Eqn(E, \mathcal{F})]\!]E$와 요약된 의미 $[\![Eqn(E, \hat{\mathcal{F}})]\!]E$는 각각 $lfp\mathcal{F}\,E$와 $lfp\hat{\mathcal{F}}\,E$와 일치한다.

요약해석 틀에 의해, $lfp\hat{\mathcal{F}}$이 $lfp\mathcal{F}$의 안전한 요약이려면, 증명할 것은

$$\alpha \circ \mathcal{F} \sqsubseteq \hat{\mathcal{F}} \circ \alpha$$

이다. 즉, 모든 프로그램 식 E에 대해서

$$\forall f : (\alpha(\mathcal{F}\,f))\,E \sqsubseteq (\hat{\mathcal{F}}(\alpha\,f))\,E.$$

여기서

$$(Exp \to Env \to Val) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (Exp \to \hat{Env} \to \hat{Val})$$

이고, 증명은 E의 각 경우별로 따지면 되는 데, 이 때

$$Env \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \hat{Env}$$

와

$$Val \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \hat{Val}$$

를 가지고 조립식으로 정의된 α가 사용된다.

분석의 구현은, 분석할 프로그램 E가 주어졌을 때 연립방정식 $Eqn(E, \hat{\mathcal{F}})$을 풀면된다. 연립방정식은 E안에 있는 모든 식 E_i의 요약 의미

$$\hat{\mathcal{V}}\, E_i \in \hat{Env} \to \hat{Val}$$

에 대한 방정식이다. 즉, 함수를 계산해 내는 것이다. 절 6.1.3에서와 같은 경우가 되겠다. 두가지 방법으로 계산할 수 있다.

- 만일 $\hat{Env}$의 원소수가 유한하다면, $\hat{Env} \to \hat{Val}$의 원소는 유한한 사이즈의 테이블이 되고, 연립방정식 $Eqn(E, \hat{\mathcal{F}})$ 즉,

$$\begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix} = \hat{\mathcal{F}}_E \begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix}$$

  ($\hat{X}_i$를 $\hat{\mathcal{V}}\, E_i$ 대신 썼다.) 의 최소 해

$$\bigsqcup_{i \in \mathbb{N}} \hat{\mathcal{F}}_E^i(\bot, \cdots, \bot) \tag{6.15}$$

  를 그대로 계산할 수 있게된다. 단, 이 때 $\hat{Val}$의 높이가 유한해야 하고, 그렇지 않을 때는 적절한 축지법(*widening operator*)을 쓰면 될 것이다.

우리의 경우, 주어진 프로그램에서 $\hat{Env}$은 유한하므로, 축지법도 필요없이, 식 (6.15)를 그대로 계산할 수 있다.

그런데 모든 $\hat{\sigma} \in \hat{Env}$ 에 대해서

$$[\![Eqn(E, \hat{\mathcal{F}})]\!]\, E_i\, \hat{\sigma}$$

을 알 필요가 있을까? 즉, 집합 (함수의 그래프)

$$\{\hat{\sigma} \mapsto [\![Eqn(E, \hat{\mathcal{F}})]\!]\, E_i\, \hat{\sigma} \mid \hat{\sigma} \in \hat{Env}\} \tag{6.16}$$

를 계산할 필요가 있을까? 하나의 요약환경 $\hat{\sigma}_0$이 있어서 주어진 프로그램 E의 모든 초기환경을 포섭하는 것이라면, 우리가 관심있는 결과는 오직 하나,

$$[\![Eqn(E, \hat{\mathcal{F}})]\!]\, E_i\, \hat{\sigma}_0$$

이 아닌가? 또, 집합 (6.16)을 계산 하는 것은 분석 시간이 너무 많이 들 수도 있고, $\hat{Env}$가 무한히 많은 원소를 가지고 있다면 아예 불가능 할 수도 있다.

- 만일 $\hat{Env}$의 원소가 무한히 많다면? 혹은 유한하지만 모든 경우를 계산할 필요가 없을 경우는? 즉, 하나의 요약환경 $\hat{\sigma}_0$이 있어서, 주어진 프로그램 E의 모든 초기환경을 포섭하는 것이라면, 우리가 관심있는 결과는 오직 하나,

  $$[\![Eqn(E, \hat{\mathcal{F}})]\!]\, E\, \hat{\sigma}_0$$

  일 것이고, 이것을 계산하는 데 필요한 만큼만 연립 방정식 $Eqn(E, \hat{\mathcal{F}})$에서 계산해 내면 될 것이다.

"필요한 만큼만 연립 방정식 $Eqn(E, \hat{\mathcal{F}})$에서 계산해"낸다? 다음과 같이

생각하면 된다. 연립방정식 $Eqn(E, \hat{\mathcal{F}})$

$$\begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix} = \hat{\mathcal{F}}_E \begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix}$$

을 확장해서, 방정식의 주인공(모르는 변수) $\hat{X}_i$를 $\hat{Env}$의 원소 수 만큼 확장해서 방정식의 주인공들이 $\hat{X}_i$에 모든 $\hat{\sigma} \in \hat{Env}$들이 각각 붙은 것들이 된 방정식을 생각해보자:

$$\begin{pmatrix} \hat{X}_0\,\sigma_0 \\ \vdots \\ \hat{X}_0\,\sigma_\omega \\ \vdots \\ \hat{X}_n\,\sigma_0 \\ \vdots \\ \hat{X}_n\,\sigma_\omega \end{pmatrix} = \hat{\mathcal{F}}_E^{\hat{Env}} \begin{pmatrix} \hat{X}_0\,\sigma_0 \\ \vdots \\ \hat{X}_0\,\sigma_\omega \\ \vdots \\ \hat{X}_n\,\sigma_0 \\ \vdots \\ \hat{X}_n\,\sigma_\omega \end{pmatrix} \tag{6.17}$$

($\hat{\mathcal{F}}_E^{\hat{Env}}$의 정의는 $\hat{\mathcal{F}}_E$로 부터 명백하다.) 위의 연립 방정식들 중에서 프로그램 E와 초기 요약환경 $\hat{\sigma}_0$에 해당하는 주인공 $\hat{X}_E\,\hat{\sigma}_0$의 방정식을 푸는데 필요한 방정식 들만을 골라내서 풀면 되겠다.

필요한 방정식들이 무한히 많아 지는 경우는? $\hat{Env}$의 원소가 무한히 많으면 그렇게 될 수 있겠다. 필요한 방정식이 $\hat{X}_i$에 대해 복수로 나오게 되는 경우, 즉, $\hat{X}_i\,\sigma_1$에 대한 방정식과 $\hat{X}_i\,\sigma_2$에 대한 방정식이 필요하게 되면, 대신에 $\hat{X}_i\,(\sigma_1 \sqcup \sigma_2)$에 대한 방정식만을 생각한다면, 계산에 넣어야 할 방정식은 $\hat{Env}$의 높이가 유한하다면 항상 유한개의 방정식만 다루게 된다. 그렇게 해서 계산된 $\hat{X}_i\,(\sigma_1 \sqcup \sigma_2)$의 해는 $\hat{X}_i\,\sigma_1$의 해와 $\hat{X}_i\,\sigma_2$의 해를 모두 포섭한다. 왜냐하면 $\hat{X}_i$가 단조(*monotonic*) 함수 이고(왜?), 방정식의 내용을 결정하는 $\hat{\mathcal{F}}$도 단조(*monotonic*) 함수 이기 때문이다.

6.1.5 요약해석 적용$_3$

이제는 명령형 언어를 생각해보자. 아래와 같이 정의될 수 있는 명령문 C가 프로그램이다.

$$
\begin{array}{rcll}
C & \to & \texttt{skip} & \\
 & | & x \texttt{ := } E & \\
 & | & \texttt{if } E \texttt{ then } C \texttt{ else } C & \\
 & | & C \texttt{ ; } C & \\
 & | & \texttt{while } E \texttt{ do } C & \\
E & \to & n & (n \in \mathbb{Z}) \\
 & | & x & \\
 & | & E \texttt{ + } E & \\
 & | & E \texttt{ < } E &
\end{array}
$$

프로그램의 의미는 일상적인 명령형 언어와 같다. 의미공간은

$$
\begin{aligned}
\mathit{Memory} &= \mathit{Loc} \xrightarrow{\text{fin}} \mathit{Value} \\
\mathit{Value} &= 2^{\mathbb{Z}} + 2^{\mathbb{B}} \\
\mathit{Loc} &= \mathit{Var} \\
\mathbb{B} &= \{T, F\}
\end{aligned}
$$

이고, 의미는 다음의 함수가 만드는 방정식의 해가 되겠다:

$$
\begin{aligned}
\mathcal{C} &\in \mathit{Cmd} \to \mathit{Memory} \to \mathit{Memory} \\
\mathcal{V} &\in \mathit{Expr} \to \mathit{Memory} \to \mathit{Value}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{C}\,\texttt{skip}\,m &= m \\
\mathcal{C}\,x\,\texttt{:=}\,E\,m &= m\{x \mapsto \mathcal{V}\,E\,m\} \\
\mathcal{C}\,\texttt{if}\,E\,\texttt{then}\,C_1\,\texttt{else}\,C_2\,m &= \mathcal{V}\,E\,m\,?\,\mathcal{C}\,C_1\,m\;:\;\mathcal{C}\,C_2\,m \\
\mathcal{C}\,C_1\,;\,C_2\,m &= \mathcal{C}\,C_2\,(\mathcal{C}\,C_1\,m) \\
\mathcal{C}\,\texttt{while}\,E\,\texttt{do}\,C\,m &= \mathcal{V}\,E\,m\,?\,\mathcal{C}\,\texttt{while}\,E\,\texttt{do}\,C\,(\mathcal{C}\,C\,m)\;:\;m \\
\mathcal{V}\,n\,m &= \{n\} \\
\mathcal{V}\,x\,m &= m\,x \\
\mathcal{V}\,E_1\,\texttt{+}\,E_2\,m &= (\mathcal{V}\,E_1\,m)\;+\;(\mathcal{V}\,E_2\,m) \\
\mathcal{V}\,E_1\,\texttt{<}\,E_2\,m &= (\mathcal{V}\,E_1\,m)\;<\;(\mathcal{V}\,E_2\,m)
\end{aligned}
$$

위의 것은 프로그램 C의 의미를 정의하지는 않는다. 프로그램에 while-문이 있으면 그 의미가 조립식으로 정의되지 않기 때문이다. 위에서 표현한 $\mathcal{C}$은 C를 가지고 만드는 방정식을 표현한 것 뿐이고, 그 방정식의 해가 프로그램 C의 의미 $\mathcal{C}\,C$가 되는 것이다. 한편, 프로그램 식 E의 의미는 명령어와는 달리 $\mathcal{V}\,E$로 정의된다. E의 부품들의 의미들로 조립식으로 정의되어 있기 때문이다.

명령문 C의 의미 방정식

$$\mathcal{C}\,C = \cdots$$

의 오른편은 위에서 표현한 재귀함수 $\mathcal{C}$의 내용을 고스란히 가지는 상위의 함수

$$\mathcal{F} : (Cmd \to Memory \to Memory) \to (Cmd \to Memory \to Memory)$$

가 정의하는 $\mathcal{F}\,\mathcal{C}\,C$가 되겠다:

$$\begin{aligned}
\mathcal{F}\,\mathcal{C}\,\texttt{skip}\,m &= m \\
\mathcal{F}\,\mathcal{C}\,x\ \texttt{:=}\ E\,m &= m\{x \mapsto \mathcal{V}\,E\,m\} \\
\mathcal{F}\,\mathcal{C}\,\texttt{if}\ E\ \texttt{then}\ C_1\ \texttt{else}\ C_2\,m &= \mathcal{V}\,E\,m\,?\,\mathcal{F}\,\mathcal{C}\,C_1\,m\ :\ \mathcal{C}\,C_2\,m \\
\mathcal{F}\,\mathcal{C}\,C_1\ ;\ C_2\,m &= \mathcal{C}\,C_2\,(\mathcal{C}\,C_1\,m) \\
\mathcal{F}\,\mathcal{C}\,\texttt{while}\ E\ \texttt{do}\ C\,m &= \mathcal{V}\,E\,m\,?\,\mathcal{C}\,\texttt{while}\ E\ \texttt{do}\ C\,(\mathcal{C}\,C\,m)\ :\ m
\end{aligned}$$

프로그램 C의 요약된 의미는 다음의 요약공간

$$\begin{aligned}
\hat{Memory} &= Loc \xrightarrow{\text{fin}} \hat{Value} \\
\hat{Value} &= \hat{\mathbb{Z}} + \hat{\mathbb{B}} \\
\hat{\mathbb{B}} &= \{\bot, T, F, \top\} \\
Loc &= Var
\end{aligned}$$

을 가지고 다음의 재귀함수의 내용으로 만들어지는 방정식의 해가 되겠다:

$$\begin{aligned}
\hat{\mathcal{C}} &\in Cmd \to \hat{Memory} \to \hat{Memory} \\
\hat{\mathcal{V}} &\in Expr \to \hat{Memory} \to \hat{Value}
\end{aligned}$$

$$
\begin{aligned}
\hat{\mathcal{C}}\ \texttt{skip}\ \hat{m} &= \hat{m} \\
\hat{\mathcal{C}}\ x\ \texttt{:=}\ E\ \hat{m} &= \hat{m}\{x \mapsto \hat{\mathcal{V}}\ E\ \hat{m}\} \\
\hat{\mathcal{C}}\ \texttt{if}\ E\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \hat{m} &= (\hat{\mathcal{C}}\ C_1\ \hat{m}) \sqcup (\hat{\mathcal{C}}\ C_2\ \hat{m}) \\
\hat{\mathcal{C}}\ C_1\ ;\ C_2\ \hat{m} &= \hat{\mathcal{C}}\ C_2 (\hat{\mathcal{C}}\ C_1\ \hat{m}) \\
\hat{\mathcal{C}}\ \texttt{while}\ E\ \texttt{do}\ C\ \hat{m} &= \hat{m} \sqcup (\hat{\mathcal{C}}\ \texttt{while}\ E\ \texttt{do}\ C\ (\hat{\mathcal{C}}\ C\ \hat{m})) \\
\hat{\mathcal{V}}\ n\ \hat{m} &= \alpha\{n\} \\
\hat{\mathcal{V}}\ x\ \hat{m} &= \hat{m}\ x \\
\hat{\mathcal{V}}\ E_1\ \texttt{+}\ E_2\ \hat{m} &= (\hat{\mathcal{V}}\ E_1\ \hat{m})\ \oplus\ (\hat{\mathcal{V}}\ E_2\ \hat{m}) \\
\hat{\mathcal{V}}\ E_1\ \texttt{<}\ E_2\ \hat{m} &= (\hat{\mathcal{V}}\ E_1\ \hat{m}) \hat{<} (\hat{\mathcal{V}}\ E_2\ \hat{m})
\end{aligned}
$$

이고,

$\hat{<}$	$\bot$	$+$	$-$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$+$	$\bot$	$\top$	F	$\top$
$-$	$\bot$	T	$\top$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$

즉, 명령문 C의 요약해석 방정식

$$\hat{\mathcal{C}}\ C = \cdots$$

의 오른편은 위에서 표현한 재귀함수 $\hat{\mathcal{C}}$의 내용을 고스란히 가지는 상위의 함수

$$\hat{\mathcal{F}} : (Cmd \to \hat{Memory} \to \hat{Memory}) \to (Cmd \to \hat{Memory} \to \hat{Memory})$$

가 정의하는 $\hat{\mathcal{F}}\,\hat{\mathcal{C}}\,C$가 되겠다:

$$\begin{aligned}
\hat{\mathcal{F}}\,\hat{\mathcal{C}}\,\texttt{skip}\,\hat{m} &= \hat{m} \\
\hat{\mathcal{F}}\,\hat{\mathcal{C}}\,x\,\texttt{:=}\,E\,\hat{m} &= \hat{m}\{x \mapsto \hat{\mathcal{V}}\,E\,\hat{m}\} \\
\hat{\mathcal{F}}\,\hat{\mathcal{C}}\,\texttt{if}\,E\,\texttt{then}\,C_1\,\texttt{else}\,C_2\,\hat{m} &= (\hat{\mathcal{C}}\,C_1\,\hat{m}) \sqcup (\hat{\mathcal{C}}\,C_2\,\hat{m}) \\
\hat{\mathcal{F}}\,\hat{\mathcal{C}}\,C_1\,;\,C_2\,\hat{m} &= \hat{\mathcal{C}}\,C_2(\hat{\mathcal{C}}\,C_1\,\hat{m}) \\
\hat{\mathcal{F}}\,\hat{\mathcal{C}}\,\texttt{while}\,E\,\texttt{do}\,C\,\hat{m} &= \hat{m} \sqcup (\hat{\mathcal{C}}\,\texttt{while}\,E\,\texttt{do}\,C\,(\hat{\mathcal{C}}\,C\,\hat{m}))
\end{aligned}$$

요약해석 틀에 의해, $lfp\hat{\mathcal{F}}$이 $lfp\mathcal{F}$의 안전한 요약이려면, 증명할 것은 아래 둘 중 하나이다:

- $\alpha \circ \mathcal{F} \sqsubseteq \hat{\mathcal{F}} \circ \alpha$, 즉, 모든 프로그램 C에 대해서

$$\forall f : (\alpha(\mathcal{F}\,f))\,C \sqsubseteq (\hat{\mathcal{F}}(\alpha\,f))\,C,$$

  혹은,

- $\alpha\,f \sqsubseteq \hat{f}$ 이면 $\alpha(\mathcal{F}\,f) \sqsubseteq \hat{\mathcal{F}}\,\hat{f}$, 즉, 모든 프로그램 C에 대해서

$$(\alpha(\mathcal{F}\,f))\,C \sqsubseteq (\hat{\mathcal{F}}\,\hat{f})\,C.$$

여기서

$$(Cmd \to Memory \to Value) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (Cmd \to \hat{Memory} \to \hat{Value})$$

이고, 증명은 C의 각 경우별로 따지면 되는 데, 이 때

$$Memory \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \hat{Memory}$$

와

$$Value \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \hat{Value}$$

를 가지고 조립식으로 정의된 α가 사용된다.

분석의 구현은, 분석할 프로그램 C가 주어졌을 때 연립방정식 $Eqn(C, \hat{\mathcal{F}})$을 풀면된다. 연립방정식은 C안에 있는 모든 명령문들 C_i와 식들 E_i의 요약 의미

$$\hat{\mathcal{C}}\, C_i \in \hat{Memory} \to \hat{Memory}$$
$$\hat{\mathcal{V}}\, E_i \in \hat{Memory} \to \hat{Value}$$

에 대한 방정식이다.

방정식의 해(프로그램의 요약된 의미)는 메모리에서 메모리로 가는 함수가 되겠고, 절 6.1.4에서와 같은 경우가 된다.

Example 27 다음 프로그램

$$x \texttt{ := } 1 \texttt{ ; while } (0 \texttt{ < } x) \texttt{ do } x \texttt{ := } x \texttt{ + } 1$$

을 생각하자. 각 부품마다 번호를 붙이자.

$$\underbrace{\underbrace{x \texttt{ := } 1}_{1} \texttt{ ; while } (0 \texttt{ < } x) \texttt{ do } \underbrace{x \texttt{ := } \underbrace{x \texttt{ + } 1}_{4}}_{3}}_{2}{}_{0}$$

각 부품의 의미 $\hat{\mathcal{C}}\, C_i$와 $\hat{\mathcal{V}}\, E_i$에 대한 방정식은 각각 $\hat{\mathcal{F}}\, C_i$와 $\hat{\mathcal{V}}\, E_i$에 의해서 아래와 같이 정의된다:

$$\begin{aligned}
\hat{\mathcal{C}}\, C_0 &= \lambda \hat{m}.\hat{\mathcal{C}}\, C_2(\hat{\mathcal{C}}\, C_1\, \hat{m}) \\
\hat{\mathcal{C}}\, C_1 &= \lambda \hat{m}.\hat{m}\{x \mapsto \hat{\mathcal{V}}\, 1\, \hat{m}\} \\
&= \lambda \hat{m}.\hat{m}\{x \mapsto \alpha\{1\}\} \\
\hat{\mathcal{C}}\, C_2 &= \lambda \hat{m}.\hat{m} \sqcup (\hat{\mathcal{C}}\, C_2\, (\hat{\mathcal{C}}\, C_3\, \hat{m})) \\
\hat{\mathcal{C}}\, C_3 &= \lambda \hat{m}.\hat{m}\{x \mapsto \hat{\mathcal{V}}\, E_4\, \hat{m}\} \\
&= \lambda \hat{m}.\hat{m}\{x \mapsto (\hat{m}\, x) \hat{+} \alpha\{1\}\}
\end{aligned}$$

방정식의 주인공 $\hat{\mathcal{C}}\,C_i$를 $\hat{X}_i$로 해서 다시 쓰면

$$\begin{aligned}
\hat{X}_0 &= \lambda\hat{m}.\hat{X}_2(\hat{X}_1\,\hat{m}) \\
\hat{X}_1 &= \lambda\hat{m}.\hat{m}\{x \mapsto +\} \\
\hat{X}_2 &= \lambda\hat{m}.\hat{m} \sqcup (\hat{X}_2\,(\hat{X}_3\,\hat{m})) \\
\hat{X}_3 &= \lambda\hat{m}.\hat{m}\{x \mapsto (\hat{m}\,x) \oplus +\}
\end{aligned}$$

위의 방정식을 푸는데, 프로그램 시작에서의 메모리의 상태가

$$\{\} \in \mathit{Var} \xrightarrow{\text{fin}} \hat{\mathit{Value}}$$

라고하면,

$$\hat{X}_0\,\{\}$$

에서 부터 "연쇄반응"을 일으키는 방정식들만 풀면 될것이다. 연쇄반응을 따라 살펴보면, 풀어야 할 방정식들은 다음과 같이 차례로 드러난다:

$$\begin{aligned}
\hat{X}_0\,\{\} &= \hat{X}_2(\hat{X}_1\,\{\}) \quad \text{따라서} \\
\hat{X}_1\,\{\} &= \{x \mapsto +\} \quad \text{따라서} \\
\hat{X}_2\,\{x \mapsto +\} &= \{x \mapsto +\} \sqcup (\hat{X}_2\,(\hat{X}_3\,\{x \mapsto +\})) \quad \text{따라서} \\
\hat{X}_3\,\{x \mapsto +\} &= \{x \mapsto +\}\{x \mapsto + \oplus +\}
\end{aligned}$$

다시 정리하면

$$\begin{aligned}
\hat{X}_0\,\{\} &= \hat{X}_2\,\{x \mapsto +\} \\
\hat{X}_1\,\{\} &= \{x \mapsto +\} \\
\hat{X}_2\,\{x \mapsto +\} &= \{x \mapsto +\} \sqcup (\hat{X}_2\,\{x \mapsto +\}) \\
\hat{X}_3\,\{x \mapsto +\} &= \{x \mapsto +\}
\end{aligned}$$

위의 방정식을 다시 쓰면 ($\hat{X}_i \star$을 $\hat{Y}_i$로)

$$\begin{aligned}
\hat{Y}_0 &= \hat{Y}_2 \\
\hat{Y}_1 &= \{x \mapsto +\} \\
\hat{Y}_2 &= \{x \mapsto +\} \sqcup \hat{Y}_2 \\
\hat{Y}_3 &= \{x \mapsto +\}
\end{aligned}$$

위의 방정식의 최소해는 고정점 계산(*fixpoint iteration*)을 통해서 다음과 같이 얻어진다:

$$\begin{aligned}
\hat{Y}_0 &= \{x \mapsto +\} \\
\hat{Y}_1 &= \{x \mapsto +\} \\
\hat{Y}_2 &= \{x \mapsto +\} \\
\hat{Y}_3 &= \{x \mapsto +\}
\end{aligned}$$

즉, 프로그램의 모든 명령문마다 실행되고 나면 x는 항상 음이아닌 정수를 가진다는 분석 결과가 나온다. □

6.1.6 프로그램 분석 알고리즘

6.1.6.1 분석 알고리즘의 입력

분석기의 입력은

1. 분석할 프로그램 C 와
2. 프로그램의 요약된 의미에 관한 방정식을 도출해 주는 $\hat{\mathcal{F}}$

$$\hat{\mathcal{F}} \in (Code \to \hat{D}) \to (Code \to \hat{D})$$

이다.

6.1.6.2 분석 알고리즘의 얼개

알고리즘의 얼개는

1. 프로그램 C에 대한 요약 해석 방정식 $Eqn(C, \hat{\mathcal{F}})$

$$\begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix} = \hat{\mathcal{F}}_C \begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix} \tag{6.18}$$

세우기. $\hat{X}_k$는 C의 부품 C_k의 요약의미($\in \hat{D}$)를 뜻하고 그 방정식

$$\hat{X}_k = \cdots$$

의 오른쪽은 $\hat{\mathcal{F}}\,\{C_i \mapsto \hat{X}_i \mid C_i \in C\}\,C_k$ 로 정의된다.

2. 방정식 (6.18) 의 해 풀기. 다음의 윗뚜껑을 계산하면 된다:

$$\bigsqcup_{i \in \mathbb{N}} \hat{\mathcal{F}}_C^i(\bot_{\hat{D}}, \cdots, \bot_{\hat{D}}) \tag{6.19}$$

$\hat{D}$의 높이가 유한하다면, 위의 것을 그대로 계산할 수 있다. $\hat{\mathcal{F}}_C$는 연속(*continuous*)이거나 단조(*monotonic*)이거나 팽창(*extensive*)하는 함수 이므로

$$(\bot_{\hat{D}}, \cdots, \bot_{\hat{D}}) \sqsubseteq \hat{\mathcal{F}}_C(\bot_{\hat{D}}, \cdots, \bot_{\hat{D}}) \sqsubseteq \hat{\mathcal{F}}_C^2(\bot_{\hat{D}}, \cdots, \bot_{\hat{D}}) \sqsubseteq \cdots$$

이고 그 끝은 유한한 n에서 멈추며 그 것

$$\hat{\mathcal{F}}_C^n(\bot_{\hat{D}}, \cdots, \bot_{\hat{D}})$$

이 식 (6.19)과 일치한다. 높이가 무한하다면, 축지법(*widening operator*)과 좁히기(*narrowing operator*) (6.1.1절)를 써서 그 윗뚜껑을 유한한 시간내에 계산하고 정교하게 다듬을 수 있다.

6.1.6.3 요약의미 공간 $\hat{D}$이 함수공간이 아닐 경우

분석 알고리즘의 내용이 그림 6.1에 있다. 가장 초보적인 고정점 반복(*fixpoint iterations*) 알고리즘이다. 프로그램 C의 분석은

$$Tabulate(\hat{\mathcal{F}}, C)$$

를 계산하는 것이다. i-번째 반복은 $\hat{\mathcal{F}}_C^i(\bot_{\hat{D}}, \cdots, \bot_{\hat{D}})$를 계산한다. 방정식을 세우고 푸는 것이 따로 있지않고 프로그램 부품 C_k의 방정식 오른편 $\hat{\mathcal{F}}\,\{C_i \mapsto \hat{X}_i \mid C_i \in C\}\,C_k$ 의 계산이 $\hat{\mathcal{F}}\,\lambda x.T(x)\,C_k$를 호출하는 것으로 구현된다. 이 알고리즘은, $\hat{D}$의 높이가 유한하다면, 유한번 반복으로 멈추게 되고 그 때의 T가 분석결과가 된다.

알고리즘의 복잡도는, 방정식의 갯수(분석할 프로그램의 부품 수, 프로그램의 크기)가 N이라고 하면, 매 반복마다 드는 최대 시간은

$$N \times \mathcal{O}(\hat{\mathcal{F}}) + N \times \mathcal{O}(\sqsubseteq)$$

```
Tabulate(F̂: (Code → D̂) → (Code → D̂), C: Code)
T, T': C → D̂;
begin
    ∀C_i of C : T(C_i) := T'(C_i) := ⊥_D̂;
    repeat
        T' := T;
        ∀C_i of C : T(C_i) := F̂ (λx.T(x)) C_i;
    until  T ⊑ T' (* no more increase *)
end
```

그림 6.1: 초보적인 프로그램 분석 알고리즘

이고, 반복의 최대 횟수는 $\hat{D}$의 높이가 된다.

$\hat{D}$의 높이가 무한하던가, 유한하더라도 빨리 알고리즘이 끝나게 하려면, 그림 6.2의 알고리즘을 사용한다. 축지법(*widening operator*) $\triangledown$을 써서 안전한 방정식의 해를 신속히 계산하고, 좁히기(*narrowing operator*) $\triangle$로 그 결과를 다듬게 된다(6.1.1절).

위의 알고리즘들(그림 6.1, 그림 6.2)이 초보적인 이유는, 불필요한 계산을 많이하기 때문이다. 매 반복마다 모든 방정식의 오른쪽을 다시 계산 한다:

$$\forall C_i \in C : T(C_i) := \hat{\mathcal{F}}\,(\lambda x.T(x))\,C_i$$

좀더 효율적이기 위해서는, 매 반복마다 다시 계산할 필요가 있는 방정식만 다시 계산하면 된다. 다시 계산할 필요가 있는 방정식은 이전 반복에서 변화가 일어난 $T(C_i)$에 영향받을 수 있는 방정식들 만이다. 매번 모든 방정식을 다시 계산할 필요는 없다. 이렇게 할일만하기 방식으로 알고리즘(*worklist algorithm*)을 정의하면 그림 6.3과 같다.

Exercise 1 그림 6.3의 알고리즘을 축지법(*widening operator*)과 좁히기(*narrowing operator*)를 쓰는 알고리즘으로 확장하라.

```
Tabulate^△_▽(F̂: (Code → D̂) → (Code → D̂), C: Code)
T, T': C → D̂;
d: D̂;
begin
    ∀C_i of C : T(C_i) := T'(C_i) := ⊥_D̂;
    repeat
        T' := T;
        ∀C_i of C :
            d := F̂ (λx.T(x)) C_i;
            if d ⋢ T'(C_i) then T(C_i) := T'(C_i) ▽ d
    until  T ⊑ T' (* no more increase *)
    repeat
        T' := T;
        ∀C_i of C : T(C_i) := T'(C_i) △ F̂ (λx.T(x)) C_i;
    until  T' ⊑ T (* no more decrease *)
end
```

그림 6.2: 초보적인 프로그램 분석 알고리즘: 축지법과 좁히기

6.1.6.4 요약의미 공간 $\hat{D}$이 함수공간일 경우

프로그램의 의미는 대게 함수공간에서 정의되는 경우가 많다: 예를들어, 환경에서 값으로의 함수로, 메모리에서 메모리의 함수로. 즉, 프로그램의 요약 의미를 결정하는 함수 $\hat{\mathcal{F}}$는 대게

$$\hat{\mathcal{F}} \in (Code \to \hat{A} \to \hat{B}) \to (Code \to \hat{A} \to \hat{B})$$

의 꼴이 된다.

이런 경우, 6.1.4절과 6.1.5절에서 논의한 데로, 실제로 필요한 정보는 프로그램의 의미인 함수 전체의 모습이 아니라 그 중 일부분으로 국한되는 경우가 많다. 즉, 프로그램 C의 의미인 함수전체 보다는 그 중에서 특정 $\hat{a} \in \hat{A}$에 대한 이미지만이 분석의 관심이 되거나 그것으로 충분한 경우가 많다. 그러한 $\hat{a}$의 예로는, 프로그램의 의미가 입출력 함수로 정의될 때 모든 가능한 입력들을 표현하는 $\top_{\hat{A}}$.

```
W : 2^Code (* worklist *)
w : Code

(* f is called when the evaluation of w (current expression) requires
   the evaluation of c (sub expression of w). *)
f(c : Code) : D̂
begin
    record that evaluation of w requires that of c;
    W := Add(W, c);
    return T(c)
end

Tabulate(F̂ : (Code → D̂) → (Code → D̂), C : Code)
T_X : Pgm → X̂;   T_Y : Pgm → Ŷ;   y : Ŷ;
begin
    ∀C_i of C : T(C_i) := T'(C_i) := ⊥_D̂;
    W := {C}
    repeat
        w := Select(W)
        y := F̂ f w
        if y ⋢ T_Y(w) then
            T_Y(w) := y
            ∀w' whose evaluation needs that of w :
                W := Add(W, w')
    until W = {}
end
```

그림 6.3: 할일만 하기 방식의 프로그램 분석 알고리즘(*worklist algorithm*)

그렇게 특정한 $\hat{a}$에 대한 프로그램 C의 분석은, 프로그램 C의 요약해석 방정식

$$\begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix} = \hat{\mathcal{F}}_C \begin{pmatrix} \hat{X}_0 \\ \vdots \\ \hat{X}_n \end{pmatrix}$$

보다는, $\hat{X}_i$를 $\hat{A}$의 원소 수 만큼 확장된 방정식

$$\begin{pmatrix} \hat{X}_0\,\hat{a}_0 \\ \vdots \\ \hat{X}_0\,\hat{a}_\omega \\ \vdots \\ \hat{X}_n\,\hat{a}_0 \\ \vdots \\ \hat{X}_n\,\hat{a}_\omega \end{pmatrix} = \hat{\mathcal{F}}_C^{\hat{A}} \begin{pmatrix} \hat{X}_0\,\hat{a}_0 \\ \vdots \\ \hat{X}_0\,\hat{a}_\omega \\ \vdots \\ \hat{X}_n\,\hat{a}_0 \\ \vdots \\ \hat{X}_n\,\hat{a}_\omega \end{pmatrix}$$

을 생각해서, 그 중에서 $\hat{X}_C\,\hat{a}$의 방정식을 푸는데 필요한 방정식들만을 푸는 방식이 된다.

그림6.4는 그렇게 구현한 초보적인 알고리즘이고, 그림6.5은 할일만 하기 방식(*worklist algorithm*)의 알고리즘이다.

Exercise 2 알고리즘들(그림6.4,6.5) 을 축지법(*widening operator*)과 좁히기(*narrowing operator*)를 사용하는 버전으로 확장하라.

$f\ (c\colon Code)\ (a\colon \hat{A}) \quad : \hat{B}$
`begin`
$\quad T_X(c) := T_X(c) \sqcup a;$
$\quad$ `return` $T_Y(c)$
`end`

$Tabulate(\hat{\mathcal{F}}\colon (Code \to \hat{A} \to \hat{B}) \to Code \to \hat{A} \to \hat{B},\ C\colon Code,\ a\colon \hat{A})$
$T_A, T'_A\colon Code \to \hat{A};$
$T_B, T'_B\colon Code \to \hat{B};$
`begin`
$\quad \forall C_i \text{ of } C : T_A(C_i) := \bot_{\hat{A}}, \quad T_B(C_i) := \bot_{\hat{B}};$
$\quad T_A(C) = a;$
$\quad$ `repeat`
$\quad\quad \langle T'_A, T'_B \rangle = \langle T_A, T_B \rangle;$
$\quad\quad \forall C_i \in C : T_B(C_i) := \hat{\mathcal{F}}\ f\ C_i\ (T_A(C_i));$
$\quad$ `until` $(T_A \sqsubseteq T'_A) \wedge (T_B \sqsubseteq T'_B)$
`end`

그림 6.4: 초보적인 프로그램 분석 알고리즘: 프로그램 의미($\in \hat{A} \to \hat{B}$)가 입출력 함수일 때

```
W : 2^Code
w : Code

(* f is called when the evaluation of w (current code) requires
   the evaluation of c (sub code of w). *)
f (c : Code) (a : Â)   : B̂
begin
    record that evaluation of w requires that of c;
    if (x ⋢ T_A(c)) then
        T_A(c) := T_A(c) ⊔ a;
        W := Add(W, c);
    return T_B(c)
end

Tabulate(F̂ : (Code → Â → B̂) → (Code → Â → B̂), C : Code, â : Â)
T_A : C → Â;    T_B : C → B̂;    y : B̂;
begin
    ∀C_i ∈ C : T_A(C_i) := ⊥_Â, T_B(C_i) := ⊥_B̂;
    T_A(C) := â;
    W := {C}
    repeat
        w := Select(W)
        y := F̂ f w T_A(w)
        if y ⋢ T_B(w) then
            T_B(w) := y
            ∀w' whose evaluation needs that of w :
                W := Add(W, w')
    until W = {}
end
```

그림 6.5: 할일만 하기 방식의 프로그램 분석 알고리즘(*worklist algorithm*): 프로그램 의미($\in \hat{A} \to \hat{B}$)가 입출력 함수일 때