자바에서의 Null을 할당할 수 있는 필드를 찾기 위한 데이터 흐름 분석

김민규, 이승희, 권용래
한국과학기술원 전자전산학과
mkkim@nks, shlee, kwon@salmosa.kaist.ac.kr

Data-flow Analysis for Finding Null-assignable Fields in Java

Minkyoon Kim, Sukhee Lee, Yong-Rae Kwon
Dept. of Electronic Engineering and Computer Science, KAIST

요 약
이 논문에서는 자바 프로그래밍 실행 중에 사용하는 점(heap)의 크기를 줄일 수 있는 데이터 흐름 분석 기법을 제안한다. 이를 고려함으로 클래식을 분석하여, 사용량 데이터에서 새로 정의되는 필드(field)들을 찾는다. 이를 위해, 이 필드의 마지막 사용 후에 null 값을 필드에 할당하면 필드가 가리키고 있었던 객체를 더욱 빠르게 회수할 수 있게 되고, 이를 통해 객체들이 처리하는 임 공간을 줄일 수 있다. 이를 알고리즘은 private 필드만 대상으로 분석을 수행한다. 우리는 클러스터 내 로직을 이 알고리즘을 확장하여 모든 필드들을 분석하고, 함수 사용량을 줄이기 위해 null을 할당하도록 바이트 코드를 자동으로 수정하는 기법을 개발하는 것이다.

1. 서론
1.1 개요
이 논문에서는 자바 프로그래밍 실행 중에 사용하는 점(heap)의 크기를 줄일 수 있는 데이터 흐름 분석 기법을 제안한다. 이를 고려함으로 클래식을 분석하여, 사용량 데이터에서 새로 정의되는 필드(field)들을 찾는다. 이를 위해, 이 필드의 마지막 사용 후에 null 값을 필드에 할당하면 필드가 가리키고 있었던 객체를 더욱 빠르게 회수할 수 있게 되고, 이를 통해 객체들이 처리하는 임 공간을 줄일 수 있다. 이를 알고리즘은 private 필드만 대상으로 분석을 수행한다. 우리는 클러스터 내 로직을 이 알고리즘을 확장하여 모든 필드들을 분석하고, 함수 사용량을 줄이기 위해 null을 할당하도록 바이트 코드를 자동으로 수정하는 기법을 개발하는 것이다.

1.2 개요
이 논문에서는 자바 프로그래밍 실행 중에 사용하는 점(heap)의 크기를 줄일 수 있는 데이터 흐름 분석 기법을 제안한다. 이를 고려함으로 클래식을 분석하여, 사용량 데이터에서 새로 정의되는 필드(field)들을 찾는다. 이를 위해, 이 필드의 마지막 사용 후에 null 값을 필드에 할당하면 필드가 가리키고 있었던 객체를 더욱 빠르게 회수할 수 있게 되고, 이를 통해 객체들이 처리하는 임 공간을 줄일 수 있다. 이를 알고리즘은 private 필드만 대상으로 분석을 수행한다. 우리는 클러스터 내 로직을 이 알고리즘을 확장하여 모든 필드들을 분석하고, 함수 사용량을 줄이기 위해 null을 할당하도록 바이트 코드를 자동으로 수정하는 기법을 개발하는 것이다.

1.3 개요
이 논문에서는 자바 프로그래밍 실행 중에 사용하는 점(heap)의 크기를 줄일 수 있는 데이터 흐름 분석 기법을 제안한다. 이를 고려함으로 클래식을 분석하여, 사용량 데이터에서 새로 정의되는 필드(field)들을 찾는다. 이를 위해, 이 필드의 마지막 사용 후에 null 값을 필드에 할당하면 필드가 가리키고 있었던 객체를 더욱 빠르게 회수할 수 있게 되고, 이를 통해 객체들이 처리하는 임 공간을 줄일 수 있다. 이를 알고리즘은 private 필드만 대상으로 분석을 수행한다. 우리는 클러스터 내 로직을 이 알고리즘을 확장하여 모든 필드들을 분석하고, 함수 사용량을 줄이기 위해 null을 할당하도록 바이트 코드를 자동으로 수정하는 기법을 개발하는 것이다.
모든 메소드는 클래스에 정의된 모든 메소드들의 집합
M(A): 클래스 A에 정의된 모든 메소드들의 집합
NPM(A): 클래스 A가 정의한 non-private 메소드들의 집합
alpath(m): 메소드 m의 모든 실행 경로들의 집합

그림 2. 알고리즘 기법에 필요한 용어 정의

조건 2.1
∀ 메소드 m∈NPM(A), ∀p∈alpath(m), 실행 경로 p에 존재하는 모든 use x를 dominate하는 def x가 존재한다.

클래스의 private 필드에 접근하거나 해당 클래스가 제공하는 non-private 메소드들을 이용해야 한다. 따라서 앞서의 클래스 A의 private 필드 x가 조건 2.1을 만족한다. 이것은 클래스 A의 non-private 메소드들이 x를 전혀 사용하지 않거나, x를 사용할 때마다 새로운 값으로 정의하여 사용한다는 것을 의미한다. 그러므로 마지막 use x 후에는 x의 값을 유지할 필요가 없다. 마지막 use x 후에는 null로 정의하여 x가 가리키던 객체의 포인터를 끊으면, GC는 x가 가리키던 객체를 회수할 가능성을 갖게 된다. 그러므로 조건 2.1을 만족하는 클래스의 예를 나타내고 있다.

2.1절에서는 앞 메소드 내에서 define되는 방법을 규칙적으로 살펴보고 2.2절에서는 메소드간의 호출을 고려하여 프로시저간(interprocedural) 분석 방법을 기술한다.

2.1. 프로시저-내(Intra-procedural) 분석

본 분석에 사용한 데이터 흐름 정보(data flow information: 이하 DFI) R은 다음과 같이 정의된다.

\[ R \subseteq P(F \times FS) \]

F는 클래스 상에 정의된 reference 타입의 private 필드들의 집합이다. FS(field status)는 각 필드들이 정의(defined)이고 사용(use)되는 방식으로 나타내고 FS=(ddu, udd, none)로 정의된다. FS의 원소들은 ddu π none π udd의 순서(ordering)를 가지고 각 원소들은 갖는 의미는 아래와 같다.

메소드 내의 입력의 포인터 또는 첫 번째로 사용하는 미지의 프로시저를 R_0이라고 하자.

- (x, ddu)∈R_0이며 메소드의 시작부에 x의 값이 변화한다는 경우
  - i) use x가シア cop 정의한다.
  - ii) 첫 번째 use x전에 def x가 존재한다.
- (x, udd)∈R_0이며 메소드의 시작부에 x의 값이 변화한다는 경우
  - i) use x가 존재한다.
  - ii) 첫 번째 use x전에 def x가 존재하지 않는다.

def x: 필드 x에 값을 할당 비트 코드
use x: 필드 x의 값을 사용하는 비트 코드
call C: 메소드 C를 호출하는 비트 코드
M(A): 클래스 A에 정의된 모든 메소드들의 집합
NPM(A): 클래스 A와 정의한 non-private 메소드들의 집합
alpath(m): 메소드 m의 모든 실행 경로들의 집합

그림 3. 조건 2.1을 만족하는 예제. 클래스 A에 정의된 메소드들의 제어 흐름 그래프. 각 메소드의 마지막 use x 후에 x=null이 추가될 수 있다.

이 문제를 위한 데이터 흐름 식(data flow equation)은 아래와 같이 정의된다.

\[ IN(s) = \bigcup_{p \in pred(s)} OUT(p) \]
\[ OUT(s) = (IN(s) - KILL(s, IN(s))) \]
\[ \cup GEN(s, IN(s)) \]

또한 IN과 OUT은 프로그램의 각 비트 코드를 DFI로 대신시킨다. S는 비트 코드들의 집합이다.

\[ IN,OUT: S \rightarrow R \]

또한 IN은 아래와 같이 초기화 시킨다.

\[ \forall s \in S, IN(s) = \{0, 1, none \} \cap \{F\} \]

GEN 함수와 KILL 함수의 각 메소드의 값은 다음과 같이 정의된다.

\[ GEN, KILL: S \rightarrow R \]

이 분석에 영향을 끼치는 basic instruction들은 필드를 접근하는 instruction인 putfield, putstatic, getfield, getstatic이 다. 이 instruction들은 대해 다음과 같이 GEN 함수와 KILL 함수를 정의한다.

\[ s \in S\text{, }s \neq x \text{의 경우.} \]

- i) s가 putfield x 또는 putstatic x일 때:
  - (none)∈R 일 경우,
    \[ GEN(s, R) = \{(x, ddu)\} \]
  - (none)∈R 일 경우,
    \[ KILL(s, R) = \{(x, none)\} \]
- ii) s가 getfield x 또는 getstatic x일 때:
  - (none)∈R 일 경우,
    \[ GEN(s, R) = \{(x, udd)\} \]
  - (none)∈R 일 경우,
    \[ KILL(s, R) = \{(x, none)\} \]
- iii) 그 밖의 경우:
  - \[ GEN(s, R) = KILL(s, R) = \phi \]

\[ \phi \]
그림 4. 프로시저간 분석의 설명을 위해 예. 메소드 C의 분석 결과에 따라 메소드 B의 분석 결과가 달라진다.

DFl $R_1$과 $R_2$가 다음과 같이 정의되었다고 하자.

$R_1 = \{ (l_1, s_{11}), (l_2, s_{12}), \ldots, (l_n, s_{1n}) \}$
$R_2 = \{ (l_1, s_{21}), (l_2, s_{22}), \ldots, (l_n, s_{2n}) \}$

$R_L R_O$는 다음과 같이 정의된다.

$R_L R_O = \{ (l_1, s_{11}), (l_2, s_{12}), \ldots, (l_n, s_{1n}) \}$
$\cup \{ (l_1, s_{21}), (l_2, s_{22}), \ldots, (l_n, s_{2n}) \}$
$\cup \{ (l_1, s_{11} L_1 s_{12}), (l_2, s_{12} L_1 s_{22}), \ldots, (l_n, s_{11} L_1 s_{2n}) \}$

$F_S = F_S$에 대해 $I_S L_1 S_T$는 위해서 연주시 $F_S$의 원소들 간의 순서에 의해 다른 값과 같다.

$\forall s \in F_S, f_S L_u u = f_S, none \neq none, f_S L_d u u = u d d$

2.2 프로시저간 (Interprocedural) 분석

정의 2.2 업의 메소드 m의 제어 흐름 그래프를 CFG(m)라고 하자. result(m)을 OUT(CFG(m))의 exit 노드로 정의한다.

그림 4에서 보면 result(C)의 값에 따라 result(B)의 값이 결정된다. 즉, $(x, u d u) \in \text{result}(C)$ 이면 $(x, u d u) \in \text{result}(B)$이고, $(x, d u u) \in \text{result}(C)$ 이면 $(x, d u u) \in \text{result}(B)$이다. 또 $(x, \text{none}) \notin \text{result}(C)$ 이면 $(x, \text{none}) \notin \text{result}(B)$이다. 이와 같이 업의 메소드 m의 result(m) 값을 구하기 위해서는 메소드 m에서 호출하는 다른 메소드들에 대한 분석 결과를 필요로 한다. 이 경우 result(m)를 구하는 방법으로는 다음 두 가지가 있다.

- 메소드 m를 분석한 후에 메소드 m에서 호출하는 다른 메소드들을 메모리에 담아, 메소드 m를 분석할 때 모든 정보가 미리 준비되어 있도록 한다.
- 메소드 m를 분석할 때 m에서 호출하는 다른 메소드들에 대한 분석 결과가 없을 경우, 해당 메소드 호출을 무시하고 분석을 계속 진행한다. 메소드 m에서 호출하는 메소드에 대한 분석 정보가 준비된 경우 메소드 m을 다시 분석한다.

첫 번째 방법은 호출 그래프(call graph)의 사이클을 고려할 수 있다. 그러므로 본 논문에서는 두 번째 방법을 사용하였다.

분석 결과에 영향을 미치는 자바 instruction들의 필드를 접근하는 instruction이 외에도 메소드 호출 instruction에 포함되어야 한다. s가 메소드 n을 호출하는 statement일 때 GEN 함수와 KILL 함수는 다음과 같이 정의된다.

그림 5. 클래스 내의 메소드들을 분석하는 worklist iteration 알고리즘

\begin{itemize}
\item i. result(m)이 존재한다면:
\begin{align*}
\text{GEN}(s, R) &= \{ (x, f_S) \mid (x, f_S) \in \text{result}(n) \land (x, \text{none}) \notin R \} \\
\text{KILL}(s, R) &= \{ (x, \text{none}) \mid (x, f_S) \in \text{result}(n) \land (x, \text{none}) \notin R \}
\end{align*}
\item ii. result(m)이 존재하지 않는다면:
\begin{align*}
\text{GEN}(s, R) &= \text{KILL}(s, R) = \emptyset
\end{align*}
\end{itemize}

메소드들을 업의 순서대로 분석할 수 있다. 어떤 메소드들의 분석 결과가 백에서는 메소드 C를 호출하는 다른 메소드들도 다시 분석하도록 하는 worklist iteration 알고리즘을 사용한다.

모든 메소드들에 대한 분석이 끝났다면, 마지막 use 후에 null을 할당할 수 있는 필드들의 집합 FinalResult는 다음과 같다.

FinalResult = \{ x \mid \phi \in NPM(\text{A}), (x, \text{null}) \in \text{result}(m) \lor (x, d u u) \in \text{result}(m) \}

3. 결론 및 향후 과제

본 논문에서는 드래그 상태에 있는 객체들을 어떻게 회수할 수 있도록 null을 할당할 수 있는 필드들을 알아내는 데이터 호출 통 분석 기법을 제안하였다. 본 논문에서 제안한 알고리즘은 private 필드만을 분석 대상으로 하고 있지만, non-private 필드도 분석 대상에 포함시키기 위한 연구가 현재 진행 중이다. 또 null을 할당하는 코드를 자동으로 바이트 코드에 삽입해 주는 방법을 사용할 계획이다.

4. 참고 문헌


