분석기 생성 시스템 Z2에서 그래픽 사용자 인터페이스의 자동 생성

어현준○ 이광근 김성훈
한국과학기술원 전산학과

Automatic Generation of Graphic User Interface on Analyzer Generating System Z2

Hyunjun Eo Kwangkeun Yi SeongHoon Kim
Dept. of Computer Science
Korea Advanced Institute of Science and Technology

요 약

시스템 Z2는 프로그램 분석기를 자동으로 만들어주는 Z1의 새로운 버젼으로써, 본 논문에서는 Z2 시스템의 그래픽 사용자 인터페이스 생성시스템에 대해서 논한다. 이 시스템에 의해 Z2의 입력으로 들어가는 분석기 명세로부터 분석기 결과를 체계적으로 보여주는 그래픽 사용자 인터페이스가 만들어 진다. 만들어지는 인터페이스는 분석할 프로그램과 분석 결과를 유기적으로 연계하여 프로그램의 분석결과를 편리하게 관찰하는 것을 도와준다.

1. 서론

시스템 Z1은 다양한 프로그래밍 언어와 여러가지 분석기법들에 대해서, 분석기 디자이너가 원하는 프로그램 분석기를 자동으로 생성하고 관리해 준다[6]. 시스템 Z1은 요약해석(Abstract Interpretation)[1]을 기반으로 만들어졌기 때문에 함수간(inter-procedural) 분석과 함수내(intra-procedural) 분석에 차이를 두지 않는다. 즉, 자료 흐름 분석에 함수 흐름 분석도(control flow graph)를 필요로 하지 않기 때문에 Standard ML[4]과 같이 함수가 다른 값과 같이 자유자재로 사용되는 언어에서의 프로그램 분석기를 쉽게 만들어낼 수 있다. 또, 래티스 프로젝션(lattice projection)[6]을 통하여 분석 비용과 정확도를 조절할 수 있는 장점이 있다.

이런 장점에도, Z1이 만들어내는 분석기를 사용해서 그 분석기의 성능을 관찰하기에는 불편한 면이 있다. 이점을 개선하여 실제로 많은 사람들이 분석기를 만들 때 사용할 수 있도록 시스템 Z2[2]가 현재 개발 중이다. 본 연구의 목표는 Z2 시스템의 일부로써, 분석할 프로그램과 분석결과를 유기적으로 연동시켜 보여주는 GUI 환경을 개발하는 것이다.

2. 시스템 Z2

2.1 전체 시스템의 구조

그림 1은 시스템 Z2의 구조를 나타낸다. Z2는 특정한 분석방법에 맞게 고안된 요약 해석기(Abstract Interpreter)를 입력으로 받아 그에 따른 C코드나 ML

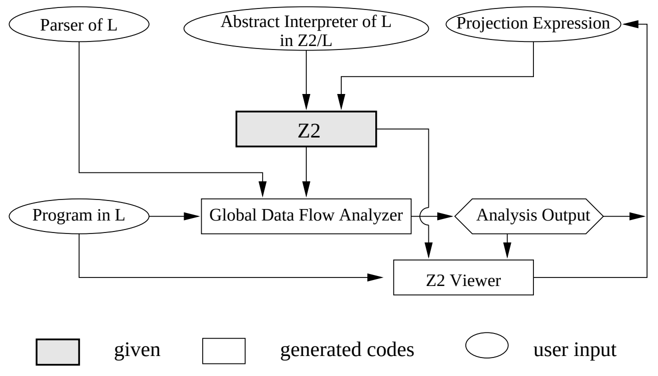

그림 1: Z2 시스템의 구조

코드를 만들어 낸다. Z2가 만들어 내는 분석기는 프로그램을 입력으로 받아서 각각의 프로그램 포인트마다 그 포인트에서 프로그램을 실행시킬때 가능한 모든 상태를 기술하는 정보를 계산하게 된다[6].

- **Z2의 입력 : 요약 해석기 (Abstract Interpreter)** 요약해석이란 프로그램이 다루는 도메인들을 요약하여 그 요약된 도메인하에서 프로그램을 요약 수행(abstract execution)하여, 그 결과가 실제 수행에 대해 어떤 정보를 줄 수 있게하는 것이다[1]. 따라서, 도메인을 어떻게 요약시킬 것인지, 그리고 프로그램의 수행을 어떻게 요약시킬 것인지가 만들어지는 분석기가 어떤 정확도로, 어느 용도에 사용될지를 결정하게 된다. Z2에서 제공하는 요약 도메인(abstract domain)들은 그림 2와 같은 래티스구조를 가진다[1]. 시스템 Z2는 이런 요약 해석기를

나열형 집합	$S = \{...\}$
정수 영역 집합	$S = [Z_l : Z_u]$
색인 집합	$S = [0 : \rho()]$
곱 집합	$S = S_1 \times ... \times S_n$
동치 집합	$S = S'$
집합 S의 리프티드 래티스	$L = S_{\perp}^{\top}$
집합 S의 파워셋 래티스	$L = 2^S$
곱 래티스	$L = L_1 \times ... \times L_n$
기본 함수 래티스	$L = L_1 \to L_2$
동치 래티스	$L = L'$

그림 2: 요약 도메인

기술하는 Z2/L이라는 언어를 제공하고 있다.

- **Z2의 출력 : 수집 분석기(Collecting Analyzer)** 시스템 Z2는 Z2/L로 기술된 요약해석기를 보고 분석기와 그에 맞는 인터페이스를 자동적으로 만들어낸다[6]. 만들어진 분석기는 수집 분석기(Collecting Analyzer)라고 불려지는 데, 각각의 프로그램 포인트마다, 그 포인트에서 프로그램이 수행 도중 가질 수 있는 모든 정보를 모은다. 어떤 프로그램 포인트가 수행되기 바로 전의 상태를 전상태(pre-state), 수행되고 난 뒤의 상태를 후상태(post-state)라 하고, Σ는 프로그램 포인트의 집합, $\overline{X}, \overline{Y}$는 각각 전상태(pre-state)와 후상태(post-state)를 나타낼 때, 분석할 언어에 대한 요약해석 $\overline{I}$는 다음과 같이 정의된다.

$$\overline{I} : \Sigma \to \overline{X} \to \overline{Y}$$

즉 수집 분석(collecting analysis)이란, 각 프로그램 포인트에서 전상태와 후상태들의 테이블을 구하는 것이다.

2.2 그래픽 사용자 인터페이스 : ZV

시스템 Z2의 사용자 인터페이스를 설계하는 데 있어서 고려할 것은 다음과 같다.

- 구현 및 이식이 용이해야 한다.
- 분석할 프로그램과 분석 결과 사이에 연관을 지어줘야 한다.
- 분석결과를 이해하기 쉬워야 한다.
- 부가비용(overhead)이 적어야 한다.

3. ZV의 구현

3.1 소스 프로그램에서 분석 결과로의 연계

Z2의 결과로 만들어진 분석기는 프로그램 포인트를 입력으로 하여 그 결과를 계산한다. Z2 시스템에서 프로그램 포인트란 구문 트리(syntax tree)의 노드를 나타내기 때문에 실제 소스 프로그램의 어느 부분에 해당하는지를 알아내기 위한 인터페이스가 필요하다. 따라서, 분석기 디자이너는 구문 분석기 부분에 다음과 같은 함수들을 추가로 정의해 주어야 한다.

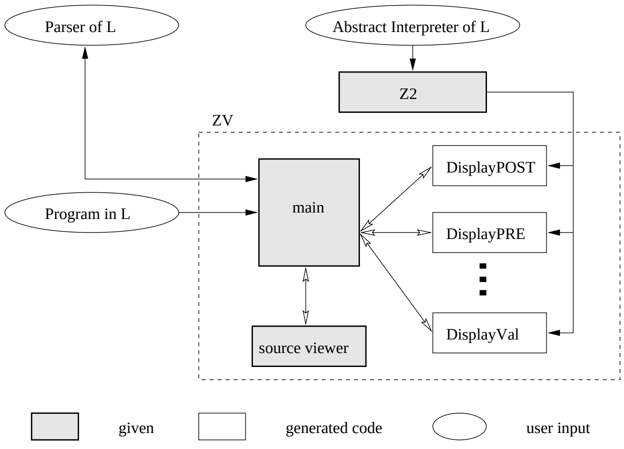

그림 3: ZV의 구조

```
TSYNTREE rangeNode(TSYNTREE program,
                   struct range *ran,
                   TSYNTREE parent);
struct range* nodeRange(TSYNTREE node);
```

`rangeNode`는 화면에 표시된 프로그램 상의 임의의 여ㅇ역으로부터 그 영역을 포함하는 가장 하위의 프로그램 포인트를 찾아주는 함수이고, `nodeRange`는 반대로 임의의 프로그램 포인트가 가지는 프로그램 상의 영역을 알려주는 함수이다. ZV는 사용자가 프로그램의 임의의 위치를 클릭하면 `rangeNode`를 호출하여 그 위치에 해당하는 프로그램 포인트를 알아내어 그 프로그램 포인트에서의 분석 결과를 보여주고, `nodeRange`를 호출하여 그 프로그램 포인트에 해당하는 영역을 화면에 표시해 준다.

3.2 분석 결과의 표시

Z2에 의해 만들어진 분석기의 분석 결과는 그림 2와 같은 래티스 구조를 가지는 요약 도메인(abstract domain)들이다. 이 래티스들은 재귀적으로 정의되어 있기 때문에 그 구조를 한번에 보여주기 힘들다. 이렇게 재귀적으로 정의될 수 있는 래티스로는 곱 도메인(product domain)과 기본 함수 도메인(atomic function domain)이 있는데, 각각의 곱 도메인, 기본 함수 도메인마다 새로운 윈도우를 만들어 그 구조를 보여주는 방법을 사용하여 분석 결과를 표시하였다.

- 집합의 표현
  - 원소 나열형 집합은 문자열로, 정수 영역 집합, 색인 집합의 원소들은 정수로 표현할 수 있다. 이 중 색인(index) 집합은 프로그램 오브젝트들(Id, 프로그램 노드 등)을 가리키는 지표이므로 색인 집합 원소로부터 프로그램 오브젝트를 알아낼 수 있는 장치를 만들어 주어야 한다.
  - 합(union) 집합과 곱(product) 집합은 수학적 표기법을 따라서 합집합은 그대로, 곱집합은 튜플로 표현하였다
- 래티스의 표현

```
Store = Loc -> Val
Loc = lift setL
Val = Z * Addr
Z = lift setZ
Addr = power setId
PRE = Store
POST = Loc * Store

setId = [0 :  NumOfIds()]
setField = [0 :  NumOfFields()]
setZ = [-99 :  99]
setRec = setId * setField
setL = setId * setRec
```

그림 4: 상수 전달 분석을 위한 S의 요약 도메인

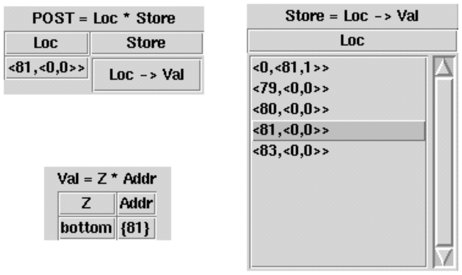

그림 5: 분석 결과의 예

– 리프티드 래티스, 파워셋 래티스와 같은 기본 래티스(atomic lattice)들은 ⊤과 ⊥을 제외하면 집합 원소들로 이루어져 있으므로, 위의 집합 표현들을 그대로 사용한다.

– 곱 래티스는 곱 집합과 마찬가지로 튜플로 생각할 수 있다. 그러나, 하위 래티스(sub lattice)가 다시 곱 래티스와 같이 기본 래티스가 아닐 경우 이를 한번에 보여주기도 어려울 뿐 아니라, 만약 한번에 보여준다고 하더라도 그렇게 보여준 결과를 해석하기 힘들게 된다. 따라서, 곱 래티스를 래티스 이름과 그 값의 테이블로 표현하고, 만약 하위 래티스가 기본 래티스가 아닐 경우에는 버튼을 만들어 그 버튼을 클릭하면 새로운 윈도우에 하위 래티스를 표현하도록 하였다.

– 기본 함수 래티스$A \rightarrow B$는 곱집합 $B^n, n = |Atoms(A)|$와 동일한 구조(isomorphic)이므로[6], 테이블로 표현할 수 있다. 그러나, n이 너무 커서 모든 테이블을 다 보여줄 수 없을 수 있기 때문에 리스트박스(listbox)[3]로 표현하였다.

그림 5는 S 언어[5]의 상수 전달(constant propagation) 분석에 대한 요약 도메인(그림 4)에 따라 만들어진 분석기가 만들어 내는 결과를 표현한 것이다. S 언어는 컴파일러 강의 시간에 사용되는 언어로 ML과 비슷한 구문구조를 가지고 있는 간단한 메모리 중심 언어(imperative language)이다.

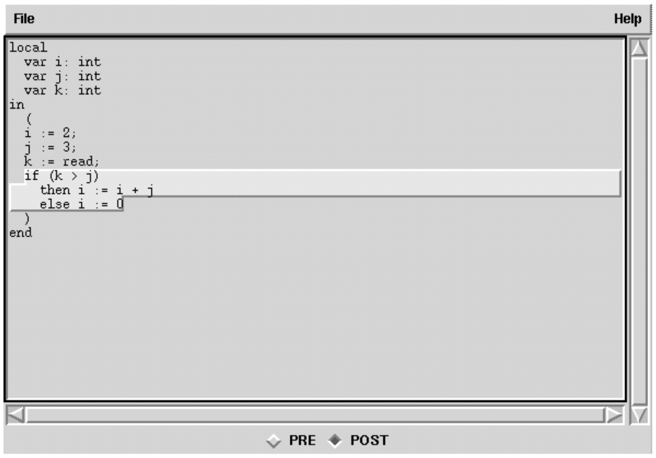

그림 6: S 언어로 만들어진 예제 프로그램

3.3 분석 결과에서 소스 프로그램으로의 연계

Z2에서 제공하는 집합은 정수 또는 나열형 뿐이므로, Id와 같은 프로그램 오브젝트를 표현하기 위해서는 색인 집합을 쓰는 방법 밖에 없다. 색인 집합은 정수로만 이루어져 있기 때문에 분석 결과로 나온 숫자가 어떤 프로그램 오브젝트인지 알기 위해서는 프로그램 오브젝트에서 색인으로의 연결 뿐만 아니라, 색인으로부터 프로그램 오브젝트로의 연결도 있어야 한다. 이 부분은 분석기 디자이너의 몫으로 색인 집합이 <S>일 때, 다음과 같은 인터페이스를 통하여 ZV와 연결할 수 있도록 하였다.

```
TSYNTREE <S>2Node(int <S>, TSYNTREE program);
```

프로그램 분석 결과를 보는 도중 사용자가 분석 결과 중 색인 집합 <S>의 한 원소를 클릭하게 되면 ZV는 `<S>2Node`를 통하여 그에 해당하는 프로그램 포인트를 알려 주고, `nodeRange`에 의해 그 영역을 보여줄 수 있게 된다.

3.4 구현 환경

그림 3에서 보듯이, ZV는 크게 도메인 정보를 이용해 Z2로부터 만들어지는 tcl/tk[3]코드와 라이브러리 부분인 tcl/tk + C코드로 이루어진다. Z2로부터 만들어지는 부분은 분석 결과를 출력하는 부분으로, 분석결과를 출력하는 부분은 도메인의 구조에 따라 달라지기 때문이다. C코드는 C로 만들어진 Z2와 tck/tk로 만들어진 ZV와의 인터페이스 부분이며, 나머지 tcl/tk코드는 프로그램의 구동기역할을 한다. tcl/tk 코드들은 tcl7.6, tk4.2를 이용하여 구현하였다.

4. 사용 예

S 언어[5]로 만들어진 간단한 예제 프로그램(그림 6)의 상수 전달(constant propagation) 분석 결과는 그림 7, 8과 같이 얻어진다.

이 결과를 보면 if-노드를 수행하기 전에는 `i = 2`, `j = 3` 이지만, if-노드를 수행하고 나면 i는 ⊤이 되어 두개 이상의 상수를 가지게 됨을 알 수 있다. 따라서,

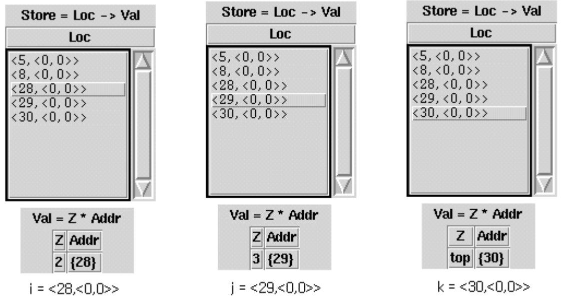

그림 7: if-노드에서의 전상태(pre-state)

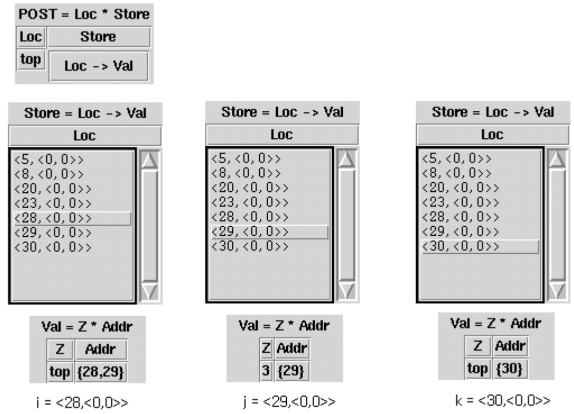

그림 8: if-노드에서의 후상태(post-state)

상수 전달 분석은 모든 프로그램을 수행하고 난 후의 후상태(post-state)에서도 여전히 단 하나의 값 만을 가지는 식(expression)들을 찾아내어 그것을 모두 상수로 치환해 줌으로써 이루어질 수 있다.

그림 7이나 그림 8에서 보듯이 `Loc <28,<0,0>>`은 변수 `i`를 의미한다. 그러나 이 `<28,<0,0>>`이 무엇을 의미하는 지를 파악하는 것은 이 분석기를 설계한 사람이라고 해도 쉽지 않은 일이다. ZV는 그림 9와 같이 이런 분석 결과와 프로그램 소스와 관계를 지어줌으로써 사용자가 분석 결과를 쉽게 해석할 수 있게 해준다. 그림에서와 같이 `<28,<0,0>>`중 `Id`를 의미하는 `28`을 클릭하면 프로그램 텍스트 상에 그에 해당하는 `i`를 `28`과 같은 색깔로 마크해 준다.

5. 결론

시스템 Z1은 프로그램 분석기를 자동으로 만들어준다. Z1은 요약해석(Abstract Interpretation)을 사용하기 때문에 고전적인 자료 흐름 분석 뿐만 아니라, 함수값이 함수의 인수로 넘기지기도 하고 결과값으로 받아지기도 하는 Standard ML과 같은 프로그래밍 언어에서의 함수 흐름 분석이나 예외상황 분석[7]도 할 수 있게 해준다.

그럼에도 시스템 Z1은 불편한 환경을 가지고 있다. Z2는 Z1의 새로운 버젼으로 좀 더 사용자에게 친근감을 주는 환경을 제공하여 실제로 컴파일러 개발에 사용되어질 수 있도록 하는 것을 목표로 한다. 현재, Z2시스템은 SUIF 프로그램 분석기를 만드는 데 사용

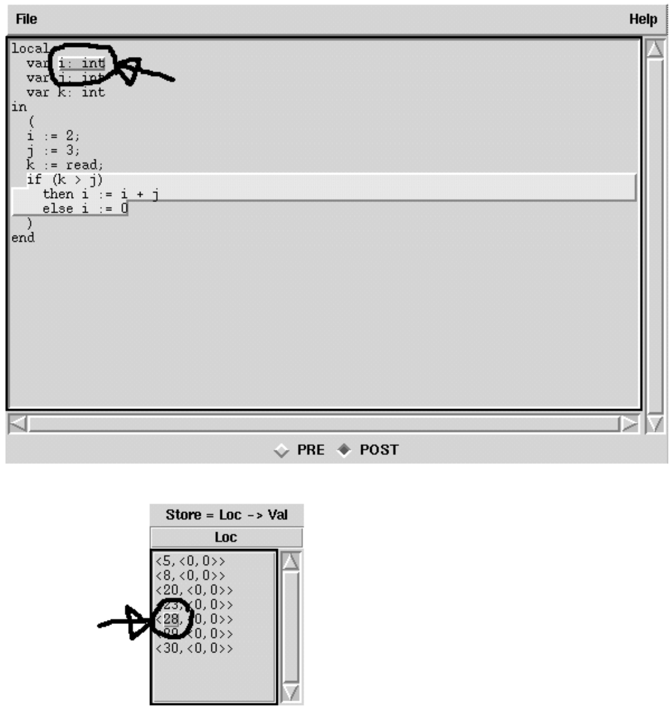

그림 9: 분석결과와 프로그램 포인트와의 관계

되고 있다[2]. 본 논문에서는 Z2 시스템이 만들어 내는 분석기의 그래픽 사용자 인터페이스를 자동으로 생성하게 하는 작업에 대해 논했다. 본 논문에서 구현된 인터페이스는 분석할 프로그램과 분석 결과를 유기적으로 연계하여 프로그램의 분석결과를 편리하게 관찰할 수 있도록 도와준다.

참고 문헌

[1] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*, 1977.

[2] Seong-Hoon Kim, Kwangkeun Yi, Hyunjun Eo, and Kwang-Moo Choe. Suif program analysis using system z2. *Proceedings of the 2nd SUIF Compiler Workshop*, August 1997.

[3] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[4] Jeffrey D. Ullman. *Elements of ML Programming*. Prentice-Hall, 1994.

[5] Kwangkeun Yi. Source language s, in revision. `http://pllab.kaist.ac.kr/~kwang/420/97/s.ps.gz`.

[6] Kwangkeun Yi and Williams Ludwell Harrison III. Automatic generation and management of interprocedural program analyses. In *Proceedings of the Annual ACM Symposium on Principles of Program ming Languages*, pages 246–259, January 1993. `http://pllab.kaist.ac.kr/~kwang/paper/popl93.ps`.

[7] Kwangkeun Yi and Sukyoung Ryu. Towards a cost-effective estimation of uncaught exceptions in sml programs. In *The 4th International Static Analysis Symposium*, September 1997.