작용 식 기반 통합 점진 해석 시스템 구축

한 정란* · 최성**

요 약

소프트웨어가 대형화되고 고도화되어 잡에 따라 프로그램 개발 환경에서 프로그래밍 생산성과 효율성을 향상시키기 위해 예비타, 컴파일러, 디버깅 및 실행을 하나의 통합 환경으로 구축하는 것이 필요하다. 이런 환경에서 중요한 도구가 되는 것이 점진 번역기이다. 본 논문에서는 소프트웨어의 재 사용성과 생산성을 향상시키기 위해, 예비타, 디버깅, 점진 해석 및 실행을 하나의 통합 환경으로 구성하여 보다 친근하고 편리하게 사용할 수 있는 사용자 인터페이스를 제공하는 소프트웨어 개발을 위한 통합 점진 해석 시스템을 구축하고자 한다. 개발 작업 언어인 IMPLOIDIPerative Language with Object) 언어를 EBNF 표기법으로 정의하고 이 언어에 대한 점진 해석기를 구현한다. 점진 해석기를 구현하기 위해 정적 의미론을 표현하는 수식 문법을 확장하여 동적 의미론을 표현할 수 있는 자료 식을 제시한다. 동적 의미 분석 방법에 의해 점진 해석을 수행하고 예비타와 디버깅을 가진 통합 점진 해석 시스템을 Lex과 Yacc를 이용해서 C 언어로 프로그래밍하고 SUN에서 X로 구현하였다. 예제 프로그램들에 점진 실행 시각을 전자 프로그램의 실행 시간과 비교했을 때 약 50% 정도의 속도 개선 효과를 기록 수 있었다.

Building of Integrated Increment Interpretation System Based on Action Equations

Jung Lan Han* · Sung Choi**

ABSTRACT

As software is large and sophisticated, in order to increase the productivity and efficiency of programs in programming development environments, it is necessary to support the integrated system that offers user interface integrated editing, compiling, debugging, and running steps. The key tool in such environments is an incremental translation. In this paper, in order to increase the productivity and reusability of software, the goal is to construct the integrated incremental interpretation system that supports friendly user interface with editor, debugger, and incremental interpreter. We define the new object-oriented language, IMPLOIDIPerative Language with Object) using EBNF notation, and construct the integrated incremental interpretation system using incremental interpreter of the language. To do so, we extend attribute grammars for specifying static semantics and present new action equations to describe the dynamic semantics. We executed the incremental interpretation by using analyzing the dynamic semantics and then implemented integrated incremental interpretation system with editor and debugger in C, Lex and Yacc using X windows on SUN. We obtain about 50% speedups in cases of incremental execution time for example programs.

키워드: 통합 프로그래밍 환경(Integrated Programming Environment), 점진 해석기(Incremental Interpreter), 점진 속성 평가(Incremental Attribute Evaluation), 종속 차트(Dependency Chart), 작용 식(Action Equation)

1. 서론

소프트웨어가 대형화되고 고도화되어 잡에 따라 소프트웨어의 유지 보수와 재사용이 점점 중요해 과제로 부각되고 있다. 소프트웨어 개발 과정에 관련된 모든 작업들을 보다 능률적으로 처리하도록 지원하는 도구에 대한 연구가 활발하게 진행되고 있다.

용용 소프트웨어나 유틸리티를 개발하기로 할 때 프로그램 개발자는 다양한 시스템 유틸리티인 에디터, 컴파일러, 런타이머 및 보다 등을 병개의 단계로 반복적으로 사용하면서 발생한 프로그램의 결과를 확인하는 과정을 통해 소프트웨어를 향상시킬 수 있도록 생산하고 소프트웨어를 개발하는데 상담한 시간을 소요하고 있다.

프로그램 개발 환경에서 프로그래밍 생산성과 효율성을 향상시키기 위해 개발 단계에서 필요한 에디터, 컴파일러, 디버깅 및 실행을 하나의 통합 환경으로 구축하여 보다 친근하고 편리하게 사용할 수 있는 사용자 인터페이스를 제공하는 통합 시스템이 필요하다. 이러한 소프트웨어 개발 환경에서 개별적인 작업을 하나의 인터페이스로 통합한 프로그램 개발 시스템이라는 통합 도구가 필요하고 이 환경에서 중요한 도구가 되는 것이 바로 점진 번역기이다. 점진 번역기는 프로그래머가 원시 프로그램을 변형할 때 작동되는 것으로 전체 프로그램을 다시 번역하지 않고 필요한 부분만을 다시 번역하는 방법으로 전체를 번역한 것과 같은 결과를 얻을 수 있다.

본 논문에서는 소프트웨어의 생산성과 실행 효율성을 향상시키기 위한 통합 작업 환경을 구축하였다.
상세히기 위해 점점 번역기를 해석 방법을 사용하여 구현 한 점점 해석기를 기반으로 하여 전체 지향 언어인 IMPLO(IMPerative Language with Object) 언어[1,2]에 대한 동합 점점 해석 시스템을 구축하고자 한다. 이 동합 시스템은 프로그램 개발 단계에서 수행되는 에디터, 인터프리팅(해석), 디버깅 및 실행을 하나의 사용자 인터페이스로 구현한 프로그램 개발 지원 시스템이다.

2. 관련 연구

소프트웨어 개발 환경에서 통합 점점 시스템의 핵심 도구인 점점 번역기를 구현하는 방법에는 해석 방식을 사용 하는 것과 컴파일 방식을 사용하는 방법이 있는데 대부분 의 경우 컴파일 기법을 사용해서 점점 번역기를 구현하고 있다.

점점 컴파일러를 구현하는 가장 간단한 방법은 언어에서 컴파일할 수 있는 최소의 단위를 결정하여 소스 코드가 변경되었을 때 이러한 가장 작은 단위를 다시 컴파일 하는 방법이다.


점점 컴파일러를 구현하기 위해 속성 문법을 사용했을 경우 문법에서 자동적으로 유도되는 속성들간의 종속 정보 를 사용해서 추이집 속성 문법에서 최소한으로 필요한 속 성이 다시 정의되어 변경된 부분에 영향받는 부분만을 다시 컴파일하게 된다. 이 방법으로 구현한 대표적 시스템으로 POE 에디터[9]가 있다.


본 연구에서는 기존의 시스템과 다르게 소프트웨어의 재 사용성을 향상시키고 개발자의 편의를 도모하기 위해 프로 그램 개발자가 프로그램을 수정할 경우, 수정된 부분에 대한 점점 스키mát 점점 과정을 수행한 후 이 부분에 영향 받는 부분들을 동적 의미 분석 방법을 통해 다시 해석하여 실행하게 한다. 기존의 연구와 다르게 동적 의미 구조를 잡은 새로운 작용식(action equation)[1,2,4,5]과 속성간의 종속성을 표현한 중속 차트(dependency chart)[1,3-5]를 통하여 수정된 명령문에 영향받는 변수들의 속성을 다시 계산하게 된다.

3. 점점 해석 방법

통합 소프트웨어 개발 환경에서 중요한 도구가 되는 것

이 점점 번역기이고 이러한 번역기는 프로그래머가 원시 프로그램을 변경할 때 적용되는 것으로 전체 프로그램을 다시 번역하지 않고 필요한 부분만을 다시 번역하는 방법이 다. 본 논문에서는 점점 번역을 해석(interpretation) 방법을 사용하여 구현하였다. 해석기는 측면지가 프로그램을 따르게 개발할 수 있도록 사용하기 위한 장점이 있다.

본 논문의 핵심 도구인 점점 해석기를 구현하기 위해 언어로 인해 필요한 새로운 작용식(action equation)[1,2,4,5]을 사용하여 동적 의미 구조를 표현하고 동적 의미 분석 방법으로 실행하게 된다. 기존의 연구와 달리 수정된 부분에 대해 변화 과정(change propagation) 과정이 복잡하게 수행되는 것을 개선하기 위해, 속성들간의 종속성을 나타내는 중속 차트(dependency chart)[1,3-5]를 만들어 수정된 부분과 그 부분에 영향받는 부분을 찾아내어 이 부분들만을 다시 실행하게 된다.

3.1 확장된 IMPLO 언어

어떤 언어에 대한 번역기를 구현하기 위해서는 언어의 동적 의미 구조를 잘 표현하는 것이 중요하다. 따라서 본 논문에서는 속성 문법을 확장하고 변화에 따라 동적 의미 구조를 표현하는 작용식(action equation)을 사용함으로써 점점 해석 방법을 사용하고자 한다. 본 점점 해석 방식

에 의해 언어로 인해 필요한 작용식을 구현하기 위해 IMPLO 언어[1,2]를 EBNF 표기법으로 각 명령문의 구문을 확장하여 정의한 다음 이 언어에 대한 점점 해석기를 구현한다.

<table>
<thead>
<tr>
<th>확장된 IMPLO 언어</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;program&gt;</strong> ::= &lt;module_list&gt;&lt;main_progran&gt;</td>
</tr>
<tr>
<td><strong>&lt;main_program&gt;</strong> ::= <strong>&lt;program&gt;</strong></td>
</tr>
<tr>
<td><strong>&lt;main_body&gt;</strong> ::= <strong>&lt;body&gt;</strong></td>
</tr>
<tr>
<td><strong>&lt;module_list&gt;</strong> ::= [<strong>&lt;module&gt;</strong>]</td>
</tr>
<tr>
<td><strong>&lt;module&gt;</strong> ::= <strong>&lt;sub_program&gt;</strong></td>
</tr>
<tr>
<td><strong>&lt;class_module&gt;</strong> ::= [<strong>&lt;modifier&gt;</strong>]&lt;class_name&gt;[<strong>&lt;derived_class&gt;</strong>]&lt;part_class&gt;</td>
</tr>
<tr>
<td><strong>&lt;derived_class&gt;</strong> ::= <strong>&lt;parent&gt;</strong></td>
</tr>
<tr>
<td><strong>&lt;part_class&gt;</strong> ::= begin</td>
</tr>
<tr>
<td><strong>&lt;statement_list&gt;</strong> ::= [<strong>&lt;statement&gt;</strong>]</td>
</tr>
<tr>
<td><strong>&lt;method_parts&gt;</strong> ::= [<strong>&lt;method_part&gt;</strong>]</td>
</tr>
<tr>
<td><strong>&lt;method_part&gt;</strong> ::= [<strong>&lt;modifier&gt;</strong>]&lt;identifier&gt;[<strong>&lt;formal_list&gt;</strong>]</td>
</tr>
<tr>
<td><strong>&lt;module_body&gt;</strong> ::= begin</td>
</tr>
<tr>
<td><strong>&lt;return_statement&gt;</strong> ::= [<strong>&lt;statement&gt;</strong>]</td>
</tr>
<tr>
<td><strong>&lt;keyword&gt;</strong> ::= procedure</td>
</tr>
<tr>
<td><strong>&lt;modifier&gt;</strong> ::= public</td>
</tr>
<tr>
<td><strong>&lt;statement&gt;</strong> ::= [<strong>&lt;in_statement&gt;</strong>]</td>
</tr>
<tr>
<td><strong>&lt;out_statement&gt;</strong> ::=</td>
</tr>
<tr>
<td><strong>&lt;class_statement&gt;</strong> ::=</td>
</tr>
<tr>
<td><strong>&lt;if_statement&gt;</strong> ::=</td>
</tr>
</tbody>
</table>
IMPL에 이어지는 명령형 언어여서 객체(object)를 다룰 수 있는 언어로 일반적인 명령형 언어의 명령문들을 배경
문, if 문, while 문, for 문 및 입출력을 다루고 있으며, 언어사용으로 할 수 있는 부분은 프로그램으로는 프로시저(pro-
cedure)와 함수(function)가 있다. 프로그램에서 메개변수(paramter)를 호출하기 위해 간호장(call-by-value)과 참조 호출(call-by-reference) 방법이 사용된다.

객체(object)를 표현하기 위해 사용자는 정의한 자료형이 클래스를 생성한다. 새로운 클래스를 생성할 때는 이 객체를 나타내는데 필요한 자료 부분과 이 객체에 사용할 수 있는 안산 부분을 정의해야 한다. IMPL 언어에서는 자료를 사용하기 위해 그 자료를 선언할 필요가 없고 배경문으로 바로 사용하게 된다. 메인에서 사용되는 동일한 이름을 자료를 참조하기 위해 'self'를 사용한다. 예를 들면 prog_name:x
는 메인 프로그램에서 사용되는 변수 x를 의미한다.

IMPL 언어의 클래스는 공용(public)과 전용(private) 파트로 구성되어간다. 전용 부분은 사용자가 이 자료를 직접 이용할 수 없고 클래스 내에 선언된 멤버 함수를 통해 사용할 수 있다. 공용 파트의 자료를 참조하기 위해서는 class_name:id와 같이 작성한다. 예를 들면 date라는 객체의 변수 x를 참조하기 위해 date.x라고 표현한다.

3.2 작용 식

어떤 언어에 대한 번역기를 구현하기 위해서는 언어의 동의어의 의미 구조를 알 필요가 있다. 따라서 본
언어에서는 수학 문법을 확장하고 변형하여 정적이고 동적

인 의미 구조를 표현하는 작용 식을 계산함으로써 본 통합
시스템의 핵심이 되는 문법 해석기를 구축하고자 한다.

작동 식에는 Execute equation, Evaluate equation, Eval-
rel equation, Eval_par equation, Wait_in equation, 및
Eval_out equation 이 있다.

Execute equation은 IMPL 언어의 각 명령문을 실행하
는 동적 명령을 표현하는 점차적인 식이다. Eval_out equa-
tion은 출력문(write 문)에 나오는 변수나 수식의 값을 계산
하는 함수적 식이다. Wait_in equation은 코보트로부터 자
료가 입력되기를 기다리며 자료가 입력되면 입력되는 자료
값을 반환하는 함수식이다. Evaluate equation은 변수의
값을 나타내는 Value(val) 속성(attribute)을 계산하기 위한
 함수식으로 계산된 숫자 값 값을 반환하는 식이다. Eval_rel
equation은 변수의 값은 명령에서 그 값을 반환하는 함수적 식이다. Eval_par equation은 완료한 수행 시 계산에서 그 수식의 값을 반환하는 함수적 식이다.

작용 식 중에서 클래스 처리를 위한 Execute equations
은 다음과 같다.

```<class 문의 Execute equation>
  • Execute [class = class_name] 
    class_name:scope = "public"
    class_name:env = class_name:env
    class_name:make_table(class_name:make, class_name:env)
    class_name:addr = current_point
  • Execute [modifier = class = class_name] 
    class_name:scope = modifier_name
    class_name:env = class_name:env
    class_name:make_table(class_name:make, class_name:env)
    class_name:addr = current_point
  • Execute [class = class_name] <derived class> 
    class_name:scope = "public"
    class_name:env = class_name:env
    class_name:parent = derived_class
    class_name:make_table(class_name:make, class_name:env)
    class_name:addr = current_point
  • Execute [modifier = class = class_name] <derived class> 
    class_name:scope = modifier_name
    class_name:env = class_name:env
    class_name:parent = derived_class
    class_name:make_table(class_name:parent, class_name:env)
    class_name:addr = current_point
  • Execute [condition <param] 
    method_name:scope = "public"
    method_name:env = id:name
    method_name:make_table(method_name:make, method_name:env)
    method_name:make_table(method_name:parent, method_name:env)
    method_name:addr = current_point
  • Execute [modifier <param>] <param>:...<param>: where n>1
```
4. 통합 정진 해석 시스템

본 논문에서 구현한 소프트웨어 개발을 위한 통합 정진 해석 시스템은 전처리된 사용자 인터페이스와 정진 해석을 수행하는 정진 해석기와 프로그램 편집을 위한 에디터와 디버그를 위한 정보를 제공하는 디버거로 구성된다. SUN에서 

X 원도우 사용자 인터페이스를 구현하고 C, Lex, 및 Yacc 

으로 통합 정진 해석 시스템을 구축하였다.

4.1 사용자 인터페이스


4.2 정진 해석기

프로그램의 실행 결과는 각 변수들의 값을 계산하고 저장하여 출력문 결과를 표시하는 출력 결과로 나타난다. 

본 논문에서 구조화는 변수의 값을 계산하여 저장하고 그 값을 출력함으로써 정확한 출력 결과를 얻을 수 있다.

정진 실행 시에 정확한 출력 결과를 얻기 위해 사용자의 구조화의 출력 결과를 토대로 각 변수들의 값을 나타내는 속성(Value attribute) 값을 계산하여 저장해야 한다. 각 변수들의 값을 저장할 때는 여러 변수들의 값을 서로 다른 위치에 저장할 수 있는데, 라인별로 각 변수들의 값을 나타내는 경우를 생각하여 각 변수들의 값을 나타내는 속성을 라인별로 값이 변한 기록을 저장한다.

정진 해석기에서 속성의 값을 정지하기 위해 정진 수행 중인 결과를 Incremental Attribute Evaluation 방법(1, 2, 4)을 사용.
용한다. 프로그램을 수정할 경우 수정된 부분만 구문 분석 하고 파생한다. 수정되었을 때 값이 변경된 변수의 Value(val) 속성에 영향받는 변수의 Value 속성을 추정함으로써 전체 프로그램을 변환한 것과 같은 결과를 얻을 수 있다. 쉽게 속성의 속성은 사용하여 수정된 변수에 영향받는 변수들을 찾아내서 그 변수가 속한 명령문을 살펴보며 정 전 해석을 수행하게 된다. 본 연구에서는 변경된 변수들에 영향받는 변수들은 찾기 위해 그림 1에서처럼 변수의 속 성간에 속성을 나타내는 종속 차트(Dependency Chart) [1,2]을 사용한다.

점진 해석기 프로그램이 수정될 때 수정된 부분에 대해 점진적으로 해석하는 과정에서 실행되는 번역 방법이고 Scmanner, Parser, Semantic analyzer, Attribute evaluator, Executor, Dependency chart generator, Tree generator, Incrementator로 구성되어 있다. 점진적으로 해석하기 위해서는 처음에 열린 파일과 수정된 후의 내용을 서로 비교하는 과정이 필요하다. 수정된 부분을 찾아내므로 파일을 로딩하고 그 부분에 대해 이하 분석과 구문 분석을 하게 된다. 프로그램의 각 문장을 Tree generator를 사용하여 랜드트리스트로 연결하였고 변경된 부분에 대해 트리 노드를 대체시키거나 삽입하거나 삭제하는 방법을 통해 변경된 부분의 변화율을 수정하고 토큰을 저장하는 비어의 내용을 역시 수정한다. 점진 해석을 통한 실행 결과를 정확하게 표시하기 위해 변수의 값이 변하게 되는 정보를 라인별로 그 값을 저장하게 된다. 변경된 부분에 대해서만 Executor를 호출함으로써 변경된 명령문을 다시 실행하여 변수 속성 정보를 받아내는 속성을 나타내는 속성과 환경을 나타내는 속성을 수정한다. 속성 차트를 통해, 변경된 부분에 대한 실행으로 인해 변화가 발생하는 영향받는 부분을 찾아내어 그 부분을 다시 실행하면 그 프로그램 전체를 실행할 필요도 없이 동일한 결과를 얻을 수 있다. 각 변수는 변수가 속해있는 라인 정보와 속성(val 속성)을 통해 저장하여 그 기록을 남겨 놓아서 동일한 변수에 대해 서로 다른 값을 가질 경우에 각 라인별로 정확한 속성 값을 가질 수 있다.

점진적으로 실행하면서 변수의 속성이 변경될 때 그 변수가 소속된 라인의 변수 속성을 변경해 주어야 정확한 결과를 얻을 수 있다. 수정되거나 삽입된 명령문에 속한 변수의 속성이 변했을 때 속성 차트를 사용해서 그 속성에 영향받는 속성을 추적하여 다시 그 값을 평가하기 위해 Attribute evaluator를 호출하게 된다. Executor로 호출하여 변경된 변수에 영향받는 속성을 포함한 명령문을 다시 실행한다. 점진 실행 후 프로그램의 결과를 출력하려면 프로그램의 모든 출력문은 수행되어 출력하여 전체를 실행한 것과 동일한 결과를 실행 화면에 표시할 수 있다.

4.3 점진 해석기 작동 알고리즘
점진적으로 해석하기 위해서는 처음에 열린 파일과 수정된 후의 내용을 서로 비교하는 과정이 필요하다. 수정된 부분을 찾아내므로 파일로 저장하고 그 부분에 대해 이하 분석과 구문 분석을 하기 위해 Scanner와 Parser를 호출하게 된다.

소 프로그램은 명령문 트리에 각 라인 단위로 명령문들의 구문이 저장되어 있다. 명령문들이 수정되면 그 명령문의
문들의 트리를 수정해야 하고 Tree generator가 호출되어 수행된다. 변경된 명령문에 속해 있는 속성들에 대해 그 속성들의 종속성을 수정해야 한다. 수정된 명령문에 속해 있는 각 속성에 대해 종속 차트(DC)를 찾아 종속성을 나타내는 포인트를 수정해야 하고 DC(종속 차트) generator를 호출하여 이러한 수정 작업을 수행하게 된다.

이러한 분석기에서 생성한 트론들을 저장하는 비버의 내용을 수정하여 의미 분석 과정을 수행하게 된다. 변경된 각 명령문에 대해 속성들을 평가하기 위해 Attribute evaluator가 호출되며 Executor에 의해 각 명령문들을 다시 실행하게 된다.

각 명령문들을 실행하면서 변수의 값을 나타내는 속성이 변경됐을 때 그 변수가 속히 있는 라인 정보와 함께 변수의 값을 저장한다. 실행 중에 변경된 각 변수의 값은 라인 정보와 함께 심볼 레이블에 모두 값이 저장되어 있으므로 라인을 확인하여 해당되는 라인의 변수 값만을 변경해야 한다.

변수의 값이 변경될 때 그 변수 값이 변경될 때 따라 값이 변경되지는 영향받는 모든 변수들을 찾아야 한다. 이를 위해 종속 차트에서 변수의 값을 나타내는 속성에 대해 후속 노드를 따라가며 값을 변경하게 될 속성을 찾아내고 이 속성이 속해있는 명령문들이 변경된 속성에 의해 영향받는 명령문이 된다. 이러한 명령문들을 다시 실행함으로써 전체를 실행하지 않더라도 전체를 실행한 것과 동일한 결과를 얻을 수 있다.

전진 해석기를 수행하기 위해 작성되는 알고리즘은 다음과 같이 기술되어 있다.

Algorithm Incremental Interpreter

1. compare opened file with modified file.
2. save modified part to X file.
3. call Scanner for X file.
4. call Parser for X file.
5. replace statement tree using Tree generator.
6. replace Dependency Chart using DC generator.
7. update buffer with toler.
8. for each statement with modified and affected attributes do
   a. call Attribute evaluator.
   b. compare old value of variable with new value.
   c. if (old value is not equal to new value) then
      i. modify new value in symbol table.
      ii. search Dependency Chart.
      iii. find changed attributes using DFS.
      iv. find affected attributes for each changed attribute.
8. od.
9. write output value in output statements.

4.4 에디터 및 디버거

edit 메뉴에는 cut, clear, copy, paste, search가 있다. 특히 search 메뉴를 누르면 실행되어 있는 search popup dialog가 (그림 3)에 나타나고 이 search popup dialog에서 찾고자 하는 문자열 입력부분과 바꾸고자 하는 문자열을 입력부분에 입력을 하면 문자열이 다이나믹하게 바뀌게 된다.

(그림 3) search 팝업 창

프로그램을 개발할 때 의미 오류 검지를 적절하게 주기 위해 디버거가 필요하다. 본 논문에서 사용한 종속 차트를 통해 변수의 값에 변화가 생기면 그 변수 값을 변함에 따라 영향받아 값이 변하는 변수들과 그 값을 표시해 주어 사용자가 의미 오류를 발견하는 작업을 도울 수 있다.

프로그램을 수정하면서 변수 값이 변화에 따라 값이 변하는 변수들을 (그림 4)에서처럼 시각적으로 보여주면 디버깅 과정이 빠르고 간단하게 수행될 수 있다.

(그림 4) 종속성 팝업 창

디버거는 "Debug" 메뉴를 선택했을 때 작동하는 것으로 종속 차트 중 종속성을 나타내는 포인트를 시각적으로 보여주게 된다.

전진 해석기를 실행하기 위해 먼저 파일을 오른.timeline 시작 Interpreters 파일에 그 내용을 읽은 다음 Interpreter 메뉴를 선택하여 프로그램 전체를 실행하게 된다. 실행된 후 프로그램의 명령문을 수정했을 경우 그 내용을 다시 저장한 후 Incrementor 메뉴의 부 메뉴 중 Compare 메뉴를 선택하여
수정 전후의 내용을 비교할 수 있다. 비교가 완료되면 Incrementor를 선택하여 수정된 부분에 대해 점검 실행을 수행하게 되고 프로그램을 실행한 결과를 검토하여 문제를 찾아보게 된다. 프로그램을 수정한 후 사용자는 Interpreter 메뉴와 Incrementor 메뉴 중 하나를 선택하여 프로그램 전체를 실행할 수도 있고 수정된 부분에 대해서만 점검 실행을 선택할 수 있다.

본 점검 해석기의 경우 사용자가 Interpreter 메뉴와 Incrementor 메뉴 중 하나를 선택하여 프로그램을 실행할 수 있으므로 오버헤드가 발생할 수 있는 문제를 수정했을 경우에는 Incrementor 메뉴보다는 Interpreter 메뉴를 선별적으로 사용할 수 있다. 디버깅하면서 점검 해석기를 수행한 화면이 (그림 5)에 표시되어 있다.

![그림 5: 디버거 실행 인터페이스](image)

### 4.4 점검 해석기의 실행 효율성 분석
점검 해석의 실행 효율성을 다양하게 분석하기 위해 내 가지 유형의 프로그램에 대해 점검 해석이 수행되는 결과를 살펴보았다. 점검 실행의 효율성을 분석하기 위해 세 가지 형태의 결과를 구성했으며, IF 문, 무프가 변경되는 경우에 나타난 유형의 프로그램 예제에 대해 점검 해석을 수행했고 점검 해석의 성능을 (표 1)에 표시하였다.

<table>
<thead>
<tr>
<th>프로그램 유형</th>
<th>배정문</th>
<th>IF 문</th>
<th>Loop 문</th>
<th>결과</th>
<th>계산 비용</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>13.1</td>
<td>15.0</td>
<td>18.2</td>
<td>20.3</td>
<td>15.3</td>
</tr>
<tr>
<td>C2</td>
<td>17.1</td>
<td>19.1</td>
<td>24.0</td>
<td>28.3</td>
<td>10.0</td>
</tr>
<tr>
<td>M1</td>
<td>6.05</td>
<td>7.05</td>
<td>8.9</td>
<td>9.7</td>
<td>3.3</td>
</tr>
<tr>
<td>M2</td>
<td>12.1</td>
<td>13.1</td>
<td>15.0</td>
<td>17.3</td>
<td>6.3</td>
</tr>
<tr>
<td>P1</td>
<td>10.0</td>
<td>12.0</td>
<td>14.0</td>
<td>16.0</td>
<td>2.0</td>
</tr>
<tr>
<td>P2</td>
<td>15.0</td>
<td>18.0</td>
<td>21.0</td>
<td>23.0</td>
<td>8.3</td>
</tr>
<tr>
<td>S1</td>
<td>11.0</td>
<td>12.1</td>
<td>14.0</td>
<td>16.0</td>
<td>2.2</td>
</tr>
<tr>
<td>S2</td>
<td>20.0</td>
<td>22.0</td>
<td>24.0</td>
<td>27.0</td>
<td>9.7</td>
</tr>
</tbody>
</table>

1. C1, C2: 계산을 뒤로 둔 메소드의 함수 규칙에 따라 수행하는 프로그램
2. M1, M2: 외부 비교, 좌표, 복수, 행렬을 위한 프로그램
3. P1, P2: 계산을 이용한 벡터를 계산하기 위한 프로그램
4. S1, S2: 함수를 이용하여 프로시저를 호출하는 프로그램

내 가지 유형의 수정에 대해 평균적으로 고려해 볼 때, 프로그램을 수정한 후 프로그램 전체를 실행하는 것과 점검 실행을 수행하는 것이 실행 효율성을 비교하여 보면 점검 수행하는 경우 50%~55% 만큼 빠르게 실행될 수 있는 양이 있다.

### 5. 결 론
본 논문에서는 소프트웨어의 생산성을 향상시키기 위해 에디터, 인터프리터, 디버깅 및 실행과정이 하나의 인터페이스 속에서 수행될 수 있는 통합 점검 해속 시스템을 구축하였다. 본 시스템에서는 사용자가 접근하기 쉽게 소프트웨어 시스템을 개발할 수 있도록 유용한 사용자 인터페이스를 제공한 것이다. 에디터와 파일 처리에 관련된 메뉴와 에디터 등을 사용하여 표준화된 인터페이스를 제공함으로써 사용자는 프로그램을 수정한 후 프로그램을 다시 실행하거나 편집기를 사용해서 점검 해속을 수행할 수 있게 되었다.

본 논문에서 제시된 도구가 되는 점검 해속기는 새로운 IMPLO 언어를 정의하고 이를 이용하여 퍼팅 해속기를 구축하였다.
신흥하였다. 수정된 부분만을 토대로 나누어 과시하여 동적 의미 복성 과정을 거치게 된다. 이 과정에서 수정된 부분에 서 값이 변한 값을 있을 경우 편소 차트를 통해 그 변수에 영향받는 변수가 숨겨 있는 명령문들을 다시 실행하게 된다. 이러한 절단 해석의 실행 결과는 전체 프로그램을 반복할 때와 같은 결과를 얻을 수 있었다. 데 가장 유명
에 프로그램을 사용해서 본 절단 해석의 실행 효율성을 검사해 보았으며 폭발적으로 고려해 볼 때 전체 프로그램
을 실행했을 경우보다 약 50% 정도의 속도 개선 효과를 얻을 수 있었다.

참고 문헌

한정란
e-mail: jhan@hyupsun.ac.kr
1985년 이화여자대학교 전자계산학과
(학사)
1987년 이화여자대학교 대학원 전자계산
학과(이학석사)
1999년 이화여자대학교 대학원 컴퓨터공학
과(박사)(공학박사)
2009년 현재 혁성대학교 경영정보학부 교수
변역기 등

최성
e-mail: ssstar@nsu.ac.kr
1983년 연세대학교 산업대학원 전자계산
학과(정보통신) 공학석사
1999년 강원대학교 대학원 컴퓨터공학과
(게임기술연구) 이학박사
1975년~1994년 기업운영 전산개발부 대리,
제주운영 전산실 실장, 한국생산
성품본 OA추진 사무국장
1994년 현재 남서울대학교 컴퓨터학과 교수
관심분야: e-Business(전자상거래/ERP/정보시스템개발), 수학
이론, 소프트웨어엔지니어링, 프로그래밍/관리, 멀티미
디어컨텐츠, 영상VR/게임, 게임개발(시나리오)