작용 식 기반 통합 점진 해석 시스템 구축

한 정 란† · 최 성††

요 약

소프트웨어가 대형화되고 고도화되어 감에 따라 프로그램 개발 환경에서 프로그래밍 생산성과 효율성을 향상시키기 위해 에디팅, 컴파일링, 디버깅 및 실행을 하나의 통합 환경으로 구축하는 것이 필요하다. 이러한 환경에서 중요한 도구가 되는 것이 점진 번역기이다. 본 논문에서는 소프트웨어의 재 사용성과 생산성을 향상시키기 위해, 에디팅, 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. 객체 지향 언어인 IMPLO(IMPerative Language with Object) 언어를 EBNF 표기법으로 정의하고 이 언어에 대한 점진 해석기를 구현한다. 점진 해석기를 구현하기 위해 정적 의미론을 표현하는 속성 문법을 확장하여 동적인 의미론을 표현할 수 있는 작용 식을 제시한다. 동적 의미 분석 방법에 의해 점진 해석을 수행하고 에디터와 디버거를 가진 통합 점진 해석 시스템을 Lex와 Yacc을 이용해서 C 언어로 프로그래밍하고 SUN에서 X로 구현하였다. 예제 프로그램들의 점진 실행 시간을 전체 프로그램의 실행 시간과 비교했을 때 약 50% 정도의 속도 개선 효과를 거둘 수 있었다.

Building of Integrated Increment Interpretation System Based on Action Equations

Jung Lan Han† · Sung Choi††

ABSTRACT

As software is large and sophisticate, 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, IMPLO(IMPerative 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 case of incremental execution time for example programs.

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

1. 서 론

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

응용 소프트웨어나 유틸리티를 개발하려고 할 때 프로그램 개발자는 다양한 시스템 유틸리티인 에디터, 컴파일러, 링커 및 로더 등을 별개의 단계로 반복적으로 사용하면서 완성된 프로그램의 결과를 출력하는 과정을 통해 소프트웨어를 비효율적으로 생산하고 소프트웨어를 개발하는데 상당한 시간을 소요하고 있다.

프로그램 개발 환경에서 프로그래밍 생산성과 효율성을 향상시키기 위해 개발 단계에서 필요한 에디팅, 컴파일링, 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 변경할 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 향

† 종신회원 : 협성대학교 경영정보학부 교수
†† 종신회원 : 남서울대학교 컴퓨터학과 교수
논문접수 : 2004년 1월 30일, 심사완료 : 2004년 4월 14일

상시키기 위해 점진 번역기를 해석 방법을 사용하여 구현한 점진 해석기를 기반으로 하여 객체 지향 언어인 IMPLO (IMPerative Language with Object) 언어[1, 2]에 대한 통합 점진 해석 시스템을 구축하고자 한다. 이 통합 시스템은 프로그램 개발 단계에서 수행되는 에디팅, 인터프리팅(해석), 디버깅 및 실행을 하나의 사용자 인터페이스로 구현한 프로그램 개발 지원 시스템이다.

2. 관련 연구

소프트웨어 개발 환경에서 통합 점진 시스템의 핵심 도구인 점진 번역기를 구현하는 방법에는 해석 방식을 사용하는 것과 컴파일 방식을 사용하는 방법이 있는데 대부분의 경우 컴파일 기법을 사용해서 점진 번역기를 구현하고 있다.

점진 컴파일러를 구현하는 가장 간단한 방법은 언어에서 컴파일할 수 있는 최소의 단위를 결정하여 소스 코드가 변경되었을 때 그러한 가장 작은 단위를 다시 컴파일 하는 방법이 있다.

컴파일 방법으로 구현된 점진 시스템들 중에서 ALOE 에디터[7]나 Gandalf 프로젝트[8]에서는 언어의 추상 구문과 관련하여 작용 루틴을 작성하여 점진 컴파일러를 구현하였다.

점진 컴파일러를 구현하기 위해 속성 문법을 사용했을 경우 문법에서 자동적으로 유도되는 속성들간의 종속 정보를 사용해서 주어진 속성 문법에서 최소한으로 필요한 속성이 다시 평가되어 변경된 부분에 영향받는 부분만을 다시 컴파일 하게 된다. 이 방법으로 구현한 대표적 시스템으로 POE 에디터[9]가 있다.

해석 방법을 사용하는 점진 해석기에서는 속성 문법을 사용하여 속성들간에 종속성을 표현하고 변화가 생겼을 때 변화된 속성에 종속하는 속성들을 점진 속성 평가 방법을 사용하여 찾아내게 된다. 이 방법으로 구현한 대표적 시스템으로 Cornell Program Synthesizer[6]가 있다.

본 연구에서는 기존의 시스템과 다르게 소프트웨어의 재사용성을 향상시키고 개발자의 편의를 도모하기 위해 프로그램 개발자가 프로그램을 수정할 경우, 수정된 부분에 대한 점진 스캐닝과 점진 파싱을 수행한 후 이 부분에 영향받는 부분들을 동적 의미 분석 방법을 통해 다시 해석하여 실행하게 된다. 기존의 연구와 다르게 동적 의미 구조를 잘 표현한 새로운 작용 식(action equation)[1, 2, 4, 5]과 속성간의 종속성을 표현한 종속 차트(dependency chart)[1, 3-5]를 통하여 수정된 명령문에 영향받는 변수들의 속성을 다시 계산하게 된다.

3. 점진 해석 방법

통합 소프트웨어 개발 환경에서 중요한 도구가 되는 것이 점진 번역기이고 이러한 번역기는 프로그래머가 원시 프로그램을 변경할 때 작동되는 것으로 전체 프로그램을 다시 번역하지 않고 필요한 부분만을 다시 번역하는 방법이다. 본 논문에서는 점진 번역을 해석(interpretation) 방법을 사용하여 구현하였다. 해석기는 초보자가 프로그램을 빠르게 개발할 수 있고 사용하기 쉬운 장점이 있다.

본 논문의 핵심 도구인 점진 해석기를 구현하려면 실제로 언어를 구현하는데 필요한 새로운 작용 식(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 파급(change propagation) 과정이 복잡하게 수행되는 점을 개선하기 위해, 속성들간의 종속성을 나타내는 종속 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 동적인 의미 구조를 잘 표현하는 것이 중요하다. 따라서 본 논문에서는 속성 문법을 확장하고 변형하여 정적이고 동적인 의미 구조를 표현하는 작용 식(action equation)을 사용함으로써 점진 해석 방법을 수행하고자 한다. 본 점진 해석 방법을 실제의 언어로 구현하여 효율성을 평가하기 위해 IMPLO 언어[1, 2]를 EBNF 표기법으로 각 명령문의 구문을 확장하여 정의한 다음 이 언어에 대한 점진 해석기를 구현한다.

```
● 확장된 IMPLO 언어
<program> ::= <module_list><main_program>
<main_program> ::= program <prog_name><main_body>
<main_body> ::= begin <statement_list> end
<module_list> ::= <module>{<module>}
<module> ::= <sub_program> | <class_module>
<class_module> ::= [<modifier>]class<class_name>
                   [<derived_class>]<method_parts><class_body>
<derived_class> ::= : parent <class_name>
<class_body> ::= begin <statement_list> end
<statement_list> ::= <statement>{<statement>}
<method_parts> ::= <method_part>{<method_part>}
<method_part> ::= [<modifier>]<identifier>(<formal_list>)
                  <module_body>
<sub_program> ::= <sub_heading><module_body>
<module_body> ::= begin <ret_statement_list> end
<ret_statement_list> ::= <statement_list>
                  | <return_statement>
<sub_heading> ::= <sub_keyword><sub_name>(<formal_list>)
<sub_keyword> ::= procedure | function
<modifier> ::= public : | private :
<statement> ::= <in_statement>
                | <out_statement>
                | <ass_statement>
                | <if_statement>
                | <for_statement>
```

```
                |<while_statement>
                |<call_statement>
                |<obj_new_statement>
<in_statement> ::= read <var_list>
<out_statement> ::= write <out_list>
<out_list> ::= <exp_title_list>{,<exp_title_list>}
<ass_statement> ::= [<env_id>]<identifier> = <exp_list>
<exp_title_list> ::= <title_list> | <exp_list>
<call_statement> ::= <module_name>(<actual_list>)
<module_name> ::= <sub_name> | [<class_id_spec>]<identifier>
<env_id> ::= <env_id_spec>::| <class_id_spec>
<class_id_spec> ::= <class_id>.
<env_id_spec> ::= <sub_name> | <prog_name>
<return_statement> ::= return <exp_list>
<if_statement> ::= if <condition> then <statement_list>
                   [else <statement_list>] fi
<for_statement> ::= for <identifier> = <exp> to <exp>
                    do <statement_list> od
<while_statement> ::= while <condition> do <statement_list> od
<condition> ::= <exp><rel_op><exp>
<obj_new_statement> ::= <class_id><identifier> =
                        new <class_id>(<formal_list>)
<class_id> ::= <identifier>
```

IMPLO 언어는 명령형 언어이면서 객체(object)를 다룰 수 있는 언어로 일반적인 명령형 언어의 명령문들인 배정문, if 문, while 문, for 문 및 입출력문을 다루고 있으며, 언어에서 호출할 수 있는 부 프로그램으로는 프로시저(procedure)와 함수(function)가 있다. 부 프로그램에서 매개변수(parameter)를 호출하기 위해 값 호출(call-by-value)과 참조 호출(call-by-reference) 방법이 사용된다.

객체(object)를 표현하기 위해 사용자가 정의한 자료형이 클래스이다. 새로운 클래스를 정의할 때는 이 객체를 나타내는데 필요한 자료 부분과 이 객체에 적용할 수 있는 연산 부분을 정의해야 한다. IMPLO 언어에서는 자료를 사용하기 위해 그 자료를 선언할 필요가 없고 배정문으로 바로 사용하면 된다. 메인에서 사용되는 동일한 이름의 자료를 참조하기 위해 '::'를 사용한다. 예를 들면 **prog_name::x**는 메인 프로그램에서 사용되는 변수 x를 의미한다.

IMPLO 언어의 클래스는 공용(public) 파트와 전용(private) 파트로 구성되어진다. 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.id**와 같이 작성한다. 예를 들면 **date**라는 객체의 변수 **x**를 참조하기 위해 **date.x**라고 표현한다.

3.2 작용 식

어떤 언어에 대한 번역기를 구현하기 위해서는 언어의 동적인 의미 구조를 잘 표현하는 것이 중요하다. 따라서 본 연구에서는 속성 문법을 확장하고 변형하여 정적이고 동적인 의미 구조를 표현하는 작용 식을 제시함으로써 본 통합 시스템의 핵심이 되는 점진 해석기를 구축하려고 한다.

작용 식에는 Execute equation, Evaluate equation, Eval_rel equation, Eval_par equation, Wait_in equation, 및 Eval_out equation 이 있다.

Execute equation은 IMPLO 언어의 각 명령문을 실행하는 동적 명세를 표현하는 절차적인 식이다. Eval_out equation은 출력문(write 문)에 나오는 변수나 수식의 값을 계산하는 함수적 식이다. Wait_in equation은 키보드로부터 자료가 입력되기를 기다렸다 자료가 입력되면 입력되는 자료 값을 반환하는 함수적 식이다. Evaluate equation은 변수의 값을 나타내는 Value(val) 속성(attribute)을 계산하기 위한 함수적 식으로 계산된 속성 값을 반환하는 식이다. Eval_rel equation은 관계 연산을 평가해서 그 관계식 값을 반환하는 함수적 식이다. Eval_par equation은 괄호를 갖는 수식을 계산해서 그 수식의 값을 반환하는 함수적 식이다.

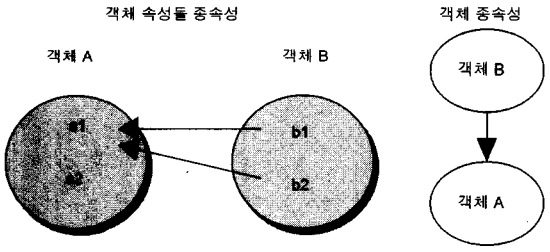
작용 식 중에서 클래스 처리를 위한 Execute equations은 다음과 같다.

<class 문의 Execute equation>

- Execute [**class** <class_name>]→
  class_name.scope ← "public"
  class_name.env ← class_name.name
  make_table(class_name.name, class_name.env)
  class_name.addr ← current_point
- Execute [<modifier> **class** <class_name>]→
  class_name.scope ← modifier.name
  class_name.env ← class_name.name
  make_table(class_name.name, class_name.env)
  class_name.addr ← current_point
- Execute [**class** <class_name><derived_class>]→
  class_name.scope ← "public"
  class_name.env ← class_name.name
  class_name.penv ← derived_class.name
  make_table(class_name.name, 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.name
  class_name.penv ← derived_class.name
  make_table(class_name.name, class_name.env)
  class_name.addr ← current_point
- Execute [<id>(<param $_1$>,<param $_2$>, ⋯ ,<param $_n$>) where n≥1]→
  method_name.scope ← "public"
  method_name.name ← id.name
  method_name.env ← method_name.name
  make_table(param $_1$.name, method_name.env)
  make_table(param $_2$.name, method_name.env)
  :
  make_table(param$_n$.name, method_name.env)
  method_name.addr ← current_point
- Execute [<modifier><id>(<param $_1$>, <param $_2$>,⋯,<param $_n$>) where n≥1]→
  method_name.scope ← modifier.name
  method_name.name ← id.name
  method_name.env ← method_name.name

```
    make_table(param 1.name, method_name.env)
    make_table(param 2.name, method_name.env)
        :
    make_table(param_n.name, method_name.env)
    method_name.addr ← current_point
• Execute [<method_name>(<param_1>, <param_2>, ..., <param_n>)
  where n ≥1] →
    param 1.env ← method_name.env
    param 2.env ← method_name.env
        :
    param n.env ← method_name.env
    param 1.val ← Eval_out[<param 1>]
    param 2.val ← Eval_out[<param 2>]
        :
    param n.val ← Eval_out[<param n>]
    return_save(current_point)
    control_transfer(method_name.addr)
• Execute [<class_id>.<method_name>(<param 1>,<param 2>,
  ···, <param_n>) where n≥1]→
  if method_name.name is class_id.name then
    param 1.env ← method_name.env
    param 2.env ← method_name.env
        :
    param n.env ← method_name.env
    param 1.val ← Eval_out[<param 1>]
    param 2.val ← Eval_out[<param 2>]
        :
    param n.val ← Eval_out[<param n>]
    return_save(current_point)
    control_transfer(method_name.addr)
  endif
• Execute [retrun <exp>] →
    exp.val ← Eval_out[<exp>]
    current_method_name.val ← exp.val
    control_transfer(current_method_name.addr)
```

IMPLO 언어에서는 객체들 간에 종속성을 표시하여야 하고 이를 위해 다음과 같은 방법으로 종속성을 나타낸다. A와 B는 객체이고 A의 속성으로 a1, a2가 있고 B의 속성으로 b1, b2이 있고 각 속성들을 다른 객체에서 사용할 수 있는 "public" 특성을 갖는다고 가정할 때 **A.a1 = B.b1 + B.b2 명령문**에서 객체 A의 속성 a1은 객체 B의 속성 b1과 b2에 종속하게 된다[1]. 즉 객체 B의 속성인 b1이나 b2의 값이 변경될 때 객체 A의 속성 a1의 값도 변하게 된다.

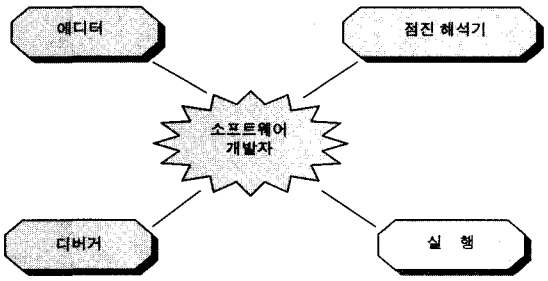


(그림 1) 객체 종속성

객체들 간의 속성들의 값이 변할 때 속성 평가를 효율적으로 수행하기 위해 객체들간의 종속성을 나타내어야 한다. 객체가 가진 속성들에서 이러한 종속성이 존재할 때 객체들간에 종속성을 (그림 1)에서처럼 표현한다.

4. 통합 점진 해석 시스템

본 논문에서 구현한 소프트웨어 개발을 위한 통합 점진 해석 시스템은 편리한 사용자 인터페이스와 점진 해석을 수행하는 점진 해석기와 프로그램 편집을 위한 에디터와 디버깅시 필요한 정보를 제공하는 디버거로 구성된다. SUN에서 X 윈도우로 사용자 인터페이스를 구현하고 C, Lex, 및 Yacc으로 통합 점진 해석 시스템을 구축하였다.



(그림 2) 통합 점진 해석 시스템

4.1 사용자 인터페이스

소프트웨어 개발을 편리하게 수행하기 위해 사용자 인터페이스에서 사용 가능한 메뉴에는 File, Edit, Interpreter, Incrementor, Debug, Help가 있다. File 메뉴는 파일을 열거나 저장하는 등의 파일 처리에 관련된 메뉴이고 Interpreter 메뉴는 전체 프로그램 파일을 해석하여 실행하는 메뉴이고 Incrementor 메뉴는 프로그램이 수정되었을 때 점진적으로 실행하기 위한 점진 해석기(점진기)에 관련된 메뉴이고 Debug 메뉴는 디버깅을 수행하기 위한 메뉴이다. 본 점진 해석기의 경우 사용자가 Interpreter 메뉴와 Incrementor 메뉴 중 하나를 선택하여 프로그램을 실행할 수 있다.

4.2 점진 해석기

프로그램의 실행 결과는 각 변수들의 값을 계산하고 저장하여 출력문의 결과를 표시하는 출력 과정으로 나타난다. 폰 노이만 구조에서는 변수의 값을 계산하여 저장하고 그 값을 출력함으로써 정확한 출력 결과를 얻을 수 있다.

점진 실행 시에 정확한 출력 결과를 얻기 위해 의미 구조에 직접적으로 영향을 주는 각 변수의 값을 나타내는 속성(Value attribute) 값을 계산하여 저장해야 한다. 각 변수들은 여러 배정문들을 통해 서로 다른 값이 배정될 수 있다. 라인별로 서로 다른 값을 나타내는 경우를 생각하여 각 변수의 값을 나타내는 속성은 라인별로 값이 변한 기록을 저장한다.

점진 해석기에서 속성들의 값을 평가하기 위해 점진 속성 평가(Incremental Attribute Evaluation) 방법[1, 2, 4]을 사

용한다. 프로그램을 수정할 경우 수정된 부분만 구문 분석하고 파싱한다. 수정되었을 때 값이 변경된 변수의 Value(val) 속성에 영향받는 변수의 Value 속성을 추적함으로써 전체 프로그램을 번역한 것과 같은 결과를 얻을 수 있다. 점진 속성 평가 방법을 사용하여 수정된 변수에 영향받는 변수들을 찾아서 그 변수가 속한 명령문을 실행하여 점진 해석을 수행하게 된다. 본 연구에서는 변경된 변수들에 영향받는 변수들을 찾기 위해 (그림 1)에서처럼 변수의 속성간에 종속성을 나타내는 종속 차트(Dependency Chart) [1, 3]를 사용한다.

점진 해석기는 프로그램이 수정될 때 수정된 부분에 대해 점진적으로 해석하는 과정에서 실행되는 번역 방법이고 Scanner, Parser, Semantic analyzer, Attribute evaluator, Executor, Dependency chart generator, Tree generator, Incrementor로 구성되어 있다. 점진적으로 해석하기 위해서는 처음에 열린 파일과 수정된 후의 내용을 서로 비교하는 과정이 필요하다. 수정된 부분을 찾아내어 파일로 저장하고 그 부분에 대해 어휘 분석과 구문 분석을 하게 된다. 이를 위해 Scanner와 Parser를 호출하게 된다.

Semantic analyzer는 의미 분석을 수행한다. 본 논문에서는 의미 분석을 위해 각 토큰을 별도의 버퍼에 저장하여 프로그램의 첫 번째 토큰부터 마지막 토큰까지 순서대로 의미 분석을 수행하고 그 오류를 검사한다. 별도의 버퍼에 각 토큰을 저장하여 의미적인 구조를 다시 검사하는 이유는 종속 차트(DC)를 효율적으로 생성하고 프로그램의 각 명령문 단위로 트리를 구성하여 변경된 부분에 대한 점진 수행을 효율적으로 실행하기 위해서이다. Semantic analyzer는 각 변수의 Value(val) 속성과 env 속성을 분석하여 형(type)을 검사하고 의미 오류(semantic error) 메시지를 준다. 점진 평가(incremental evaluation)를 수행하기 위해 DC(Dependency Chart) generator를 호출하여 종속 차트(dependency chart)를 생성하고 Tree generator를 호출하여 각 명령문 별로 구문이 저장되는 트리를 생성한다.

DC(Dependency Chart) generator는 종속 차트(dependency chart)를 생성하는 종속 차트 생성기이다. 종속 차트는 종속성을 나타내는 필드와 조건 속성을 나타내는 CON 필드와 필수 속성을 구분 짓기 위한 FLAG 필드가 들어간다. 속성을 빠르게 검색하기 위해 속성들의 테이블인 AT(Attribute Table)가 구성된다[1, 2].

Tree generator는 IMPLO 소스 언어를 각 라인 단위로 명령문의 구문이 저장되는 트리를 생성한다. 각 명령문이 소속된 라인을 나타내기 위해 라인 정보가 들어가고 각 라인에 있는 명령문의 구문이 들어가고 다음 명령문을 연결하는 링크가 포함된다.

Attribute evaluator는 의미 분석을 통해 각 변수의 값을 나타내는 Value(val) 속성과 그 외의 여러 속성들을 평가하는 작업을 수행한다.

Executor는 작용 식(action equation)에 의해 명세된 의미 구조에 따라 각 명령문단위로 명령문을 실제로 수행하여 IMPLO 언어의 실행 결과를 화면에 표시한다. 각 명령문 단위로 Execute action에 정의된 동적 의미 구조대로 각 명령문을 수행하게 된다.

Incrementor는 IMPLO 언어의 입력 소스를 수정했을 경우 수정된 명령문만을 어휘 분석하고 구문 분석하여 Tree generator에 의해 생성된 트리의 노드를 대치시키거나 삽입하거나 삭제한다. 수정된 명령문만을 데이터 파일에 저장하여 어휘 분석과 구문 분석을 하게 된다. 프로그램의 각 문장들을 Tree generator를 사용하여 링크트 리스트로 연결하였고 변경된 부분에 대해 트리의 노드를 대치시키거나 삽입하거나 삭제하는 방법을 통해 변경된 부분의 변화를 수정하고 토큰을 저장하는 버퍼의 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 병령문을 다시 실행하여 변수 속성 정보 즉 값을 나타내는 속성과 환경을 나타내는 속성을 수정한다. 종속 차트를 통해, 변경된 부분에 대한 실행으로 인해 변화가 발생하는 영향받는 부분을 찾아내어 그 부분을 다시 실행하면 프로그램 전체를 실행하지 않더라도 동일한 결과를 얻을 수 있다. 각 변수는 변수가 속해있는 라인 정보와 속성(val 속성)을 함께 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 속성만을 변경해 주어야 정확한 결과를 얻을 수 있다. 수정되거나 삽입된 명령문에 속한 변수의 속성이 변했을 때 종속 차트를 사용해서 그 속성에 영향받는 속성을 추적하여 다시 그 값을 평가하기 위해 Attribute evaluator를 호출하게 된다. Executor를 호출하여 변경된 변수에 영향받는 속성을 포함한 명령문을 다시 실행한다. 점진 실행 후 프로그램의 결과를 출력하려면 프로그램의 모든 출력문을 수행하여 출력하면 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 찾아내어 파일로 저장하고 그 부분에 대해 어휘 분석과 구문 분석을 하기 위해 Scanner와 Parser를 호출하게 된다.

소스 프로그램은 명령문 트리에 각 라인 단위로 명령문들의 구문이 저장되어 있다. 명령문들이 수정되면 그 명령

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 포인터를 수정해야 하고 DC(종속 차트) generator를 호출하여 이러한 수정 작업을 수행하게 된다.

어휘 분석기에서 생성한 토큰들을 저장하는 버퍼의 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 변했을 때 그 변수가 속해 있는 라인 정보와 함께 변수의 값을 저장한다. 실행 중에 변하는 각 변수의 값은 라인 정보와 함께 심볼 테이블에 모든 값이 저장되어 있으므로 라인을 확인하여 해당되는 라인의 변수 값만을 변경해야 한다.

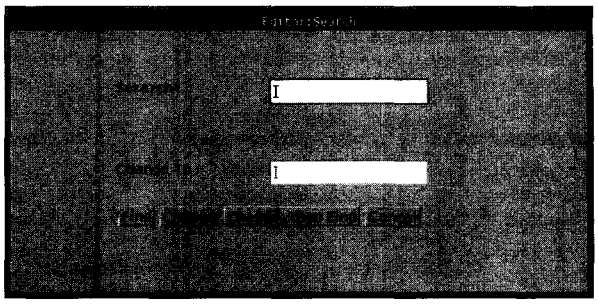
변수의 값이 변했을 때 그 변수 값이 변함에 따라 값이 변경되어지는 영향받는 모든 변수들을 찾아야 한다. 이를 위해 종속 차트에서 변수의 값을 나타내는 속성에 대해 후속 노드를 따라가며 값이 변하게 될 속성들을 찾아내고 이 속성들이 속해있는 명령문들이 변경된 속성에 의해 영향받는 명령문이 된다. 이러한 명령문들을 다시 실행함으로써 전체를 실행하지 않더라도 전체를 실행한 것과 동일한 결과를 얻을 수 있다.

점진 해석기를 수행하기 위해 작동되는 알고리즘이 다음과 같이 기술되어 있다

```
Algorithm Incremental Interpreter
  compare opened file with modified file.
  save modified part to X file.
  call Scanner for X file.
  call Parser for X file.
  replace statement tree using Tree generator.
  replace Dependency Chart using DC generator.
  update buffer with token.
  for each statement with modified and affected attributes do
    call Attribute evaluator.
    compare old value of variable with new value.
    if (old value is not equal to new value) then
       modify new value in symbol table.
       search Dependency Chart.
       find changed attributes using DFS
       find affected attributes for each changed attribute
    fi
    call Executor
  od
  write output value in output statements.
```

4.4 에디터 및 디버거

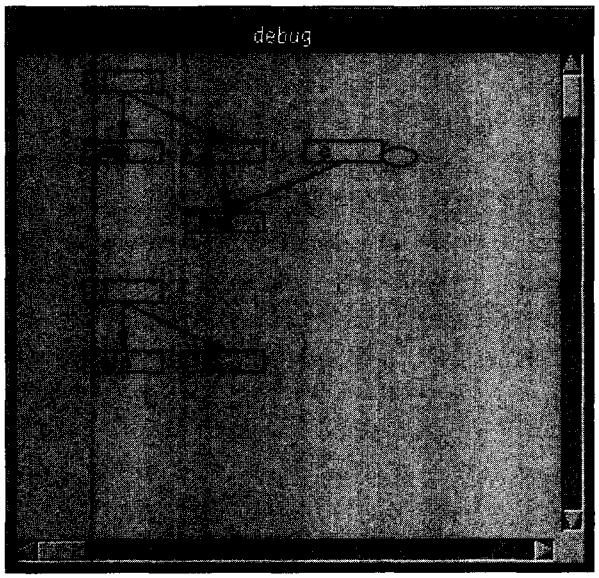
edit 메뉴에는 cut, clear, copy, paste, search가 있다. 특히 search 메뉴를 누르면 실행이 되는 search popup dialog가 (그림 3)에 나와 있고 이 search popup dialog에서 찾고자하는 문자열 입력부분과 바꾸고자 하는 문자열을 입력부분에 입력을 하면 문자열이 다이나믹하게 바뀌게 된다.



(그림 3) search 팝업 셸

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

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



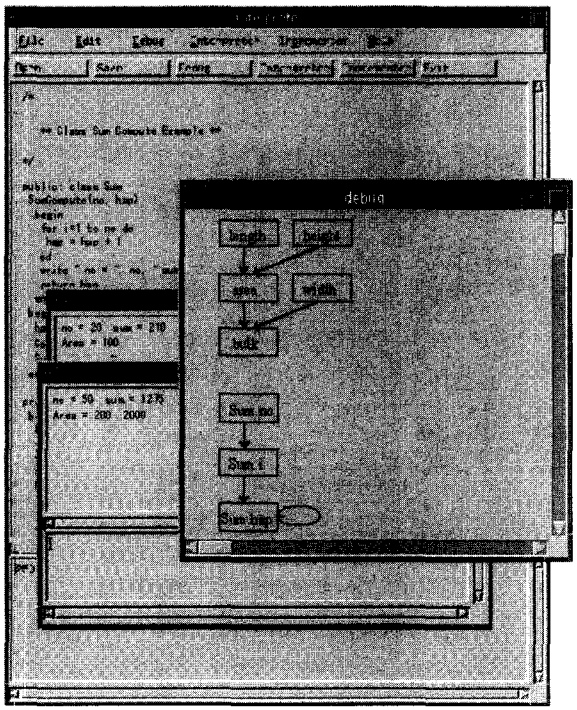
(그림 4) 종속성 팝업 셸

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

점진 해석을 실행하기 위해 먼저 파일을 오픈하여 인터페이스 화면에 그 내용을 올린 다음 Interpreter 메뉴를 선택하여 프로그램 전체를 실행하게 된다. 실행한 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 점진 실행을 수행하게 되고 프로그램을 실행한 결과를 팝업 셀에 출력하게 된다. 프로그램을 수정한 후 사용자는 Interpreter 메뉴와 Incrementor 메뉴 중 하나를 선택하여 프로그램 전체를 실행할 수도 있고 수정된 부분에 대해서만 점진 실행을 선택할 수 있다.

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



(그림 5) 디버거 실행 인터페이스

4.4 점진 해석기의 실행 효율성 분석

점진 해석의 실행 효율성을 다양하게 분석하기 위해 네 가지 유형의 프로그램에 대해 점진 해석이 수행되는 결과를 살펴보았다. 점진 실행의 효율성을 분석하기 위해 세 가지 형태의 수정, 즉 배정문들, IF 문, 루프이 변경되는 경우에 네 가지 유형의 프로그램 예제에 대해 점진 해석을 실행해 보았고 점진 해석기의 성능을 <표 1>에 표시하였다.

<표 1>을 통해 알 수 있듯이 배정문들이나 IF문을 수정했을 경우는 수정된 부분만을 번역하여 결과를 얻는 시간은 프로그램 전체를 실행한 시간에 비하면 0.5~1ms로 아주 적은 시간이다. 그러나, 루프가 수정된 경우는 수정된 루프 속에 재실행할 필요가 있는 명령문들의 수에 의해 점진 해석기의 실행 효율성이 좌우된다. 수정된 명령문에 의해 영향받는 명령문들이 많은 경우 그들에 대해 점진 해석을 수행하기 위해 종속 차트를 추적하여야 하므로 점진 평가에 대한 오버헤드가 추가로 생기게 된다. 루프를 수정했을 경우는 점진 평가에 대한 오버헤드까지 포함되기 때문에 전체 프로그램을 실행한 시간보다 더 많은 시간이 소요된다. 따라서, 사용자는 선별적으로 점진 해석을 수행할지 전체 프로그램을 실행할 지 결정하여 Interpreter 메뉴나 Incrementor 메뉴로 실행할 수 있다.

<표 1> 네 가지 예제 프로그램의 실행 시간 단위 : ms

프로그램 유형	배정문		IF 문		Loop 문		평 균		개 선 비 율
	전체	점진	전체	점진	전체	점진	전체	점진	
C1	13	1	15	1	18	20	15.3	7.3	52%
C2	17	1	19	1	24	28	20	10	50%
M1	6	0.5	7	0.5	8	9	7	3.3	52.9%
M2	12	1	13	1	15	17	13.3	6.3	52.6%
P1	10	0.5	12	0.5	14	16	12	5.67	52.8%
P2	15	1	18	1	21	23	18	8.3	53.7%
S1	11	1	12	1	14	16	12.3	6	51%
S2	20	1	22	1	24	27	22	9.7	55.9%

C1, C2 : 객체를 이용해 행렬의 합과 곱을 구하는 프로그램
M1, M2 : 최대값, 최소값, 합과 평균을 구하는 프로그램
P1, P2 : 객체를 이용해 급료를 계산하는 프로그램
S1, S2 : 합계를 구하는 프로시저를 호출하는 프로그램

네 가지 유형의 수정에 대해 평균적으로 고려해 볼 때, 프로그램을 수정한 후 프로그램 전체를 실행하는 것과 점진 실행을 수행하는 것의 실행 효율성을 비교하여 보면 점진 수행하는 경우 50%~55.9% 만큼 빠르게 실행되는 것을 알 수 있다.

5. 결 론

본 논문에서는 소프트웨어의 생산성을 향상시키기 위해 에디팅, 인터프리팅, 디버깅 및 실행과정이 하나의 인터페이스에서 수행될 수 있는 통합 점진 해석 시스템을 구축하였다. 본 시스템에서는 사용자가 편리하게 소프트웨어 시스템을 개발할 수 있도록 유용한 사용자 인터페이스를 제공해 준다. 에디터와 파일 처리에 관련된 메뉴와 에디팅을 위한 메뉴를 제공하고 디버깅시 필요한 정보를 주는 디버거는 수정된 변수 값에 영향받아 변하는 변수들을 시각적으로 보여주게 된다. 특히, 점진 해석을 수행하는 점진기와 전체 프로그램을 수행할 수 있는 인터프리터 메뉴를 제공하여 주어, 사용자는 프로그램을 수정한 후 전체 프로그램을 다시 실행하거나 점진기를 사용해서 점진 해석을 수행할 지 선택할 수 있다.

본 논문에서 핵심 도구가 되는 점진 해석기는 새로운 IMPLO 언어를 정의하고 이 언어에 대한 점진 해석기를 구

현하였다. 수정된 부분만을 토큰으로 나누어 파싱하여 동적 의미 분석 과정을 거치게 된다. 이 과정에서 수정된 부분에서 값이 변한 변수가 있을 경우 종속 차트를 통해 그 변수에 영향받는 변수가 속해 있는 명령문들을 다시 실행하게 된다. 이러한 점진 해석기의 실행 결과는 전체 프로그램을 번역했을 때와 같은 결과를 얻을 수 있었다. 네 가지 유형의 프로그램을 사용해서 본 점진 해석기의 실행 효율성을 검사해 보았으며 평균적으로 고려해 볼 때 전체 프로그램을 실행했을 경우 보다 약 50% 정도의 속도 개선 효과를 얻을 수 있었다.

참 고 문 헌

[1] 한정란, "작용 식 기반 점진 해석", Ph. D Thesis 이화여대, 1999.

[2] 한정란, 이기호, "작용 식 기반 점진 해석기", 정보과학회논문지, 제26권 제8호, pp.1018-1027, 1999.

[3] 이기호, 한정란, "순환 속성 문법의 효율적 점진 평가 기법", 정보과학회논문지, 제21권 제6호, pp.1116-1126, 1994.

[4] 한정란, 이기호, "종속 차트를 사용한 점진 해석 시스템 구축", 정보과학회 춘계 학술발표논문집, 제26권 제1호, pp.87-89, 1999.

[5] 이기호, 한정란, "점진 해석기 기반 소프트웨어 개발 통합 시스템 구축", 핵심전문연구보고서, 한국과학재단, 2000.

[6] T. Teitelbaum and T. Reps, "The Cornell Program Synthesizes : A Syntax-directed Environment," Communication ACM, Vol.24, No.9, pp.563-573, 1981.

[7] Raul Medina Rora and David S. Notkim, "ALOE users' and implementers' guide," Carnegie-mellon Computer Science Depart. Research Report CS-81-145, 1981.

[8] A. N. Habermann, "The Gandalf Research project," Computer Science research Review Carnegie-Mellon University, 1979.

[9] Charles N. F., Greg J. and Jon M., "An Introduction to Editor Allen Poe," Univ. Wisconsin-Madison TR 451, 1981.

[10] John F. Beetem and Anne F. Beetem, "Incremental Scanning and Parsing With Galaxy," IEEE Transactions on Software Engineering, Vol.17, No.7, pp.641-651, 1991.

[11] Roger Hoover, "Alphonse : Incremental Computation as a Programmer Abstraction," ACM SIGPLAN Notices, pp. 261-272 1992.

한 정 란

e-mail : jlhan@hyupsung.ac.kr

1985년 이화여자대학교 전자계산학과 (학사)

1987년 이화여자대학교 대학원 전자계산학과(이학석사)

1999년 이화여자대학교 대학원 컴퓨터공학과 PL전공 (공학박사)

1999년~현재 협성대학교 경영정보학부 교수

관심분야 : e-Commerce, XML, e-Business, e-CRM, 점진 번역기 등

최 성

e-mail : sstar@nsu.ac.kr

1983년 연세대학교 산업대학원 전자계산학과(정보통신) 공학석사

1999년 강원대학교 대학원 컴퓨터과학과 (과학기술연구) 이학박사

1975년~1994년 기업은행 전산개발부 대리, 제주은행 전산실 실장, 한국생산성본부 OA추진사무국장

1994년~현재 남서울 대학교 컴퓨터학과 교수

관심분야 : e-Business(전자상거래/ERP/),정보시스템개발, 수학이론, 소프트웨어엔지니어링, 프로젝트관리, 멀티미디어컨텐츠, 영상VR게임, 게임개발(시나리오)