nML 컴파일러 시스템 (status report)*

이광근 이욱세 어현준 김정택 최웅식 류석영 강현구 서선애 장성순 김범식†
한국과학기술원 프로그램 분석 시스템 연구단

The nML Compiler System (status report)

Kwangkeun Yi, Oukseh Lee, Hyunjun Eo, Jungtaek Kim, Woongsik Choi,
Sukyoung Ryu, Hyun-Goo Kang, Sunae Seo, Sungsoon Jang, and Bomshik Kim
Research On Program Analysis System, KAIST

요 약

상위의 타입 구조를 갖춘 (higher-order and typed) nML을 고안하고 그 컴파일러를 개발하였다. nML은 프로그래밍 언어 분야의 중요한 성과들을 충실히 실현한 언어로서, Standard ML과 Objective Caml의 장점과 실제 프로그래밍 세계의 관성을 염두에 두고 정의되었다. nML 컴파일러의 실용성과 안전성은 이미 여러 응용 시스템 소프트웨어의 개발로 실험되고 있다. 이 전통적인 컴파일러 시스템에 본 연구단의 프로그램 분석기술들이 장착되면서, 실시간 안전성 검증이 가능하고, 작고, 자발적인 적응력 있는 코드를 만들어내는 컴파일러로 발전하는 발판이 마련되었다.

1 동기

상위의 타입 구조를 갖춘 (higher-order and typed) nML을 고안하고 그 컴파일러를 개발하였다. 왜냐하면:

- 넷트웍 환경에 적합한 ML 컴파일러가 필요하다. 기존의 ML 컴파일러들(SML/NJ[2], OCaml[9])은 미래의 넷트웍 컴퓨팅 환경을 염두에 두고 만들어지지 않았다. nML 컴파일러가 만들어 내는 코드는, (1) 넷트웍을 통해 전달된 코드가 안전하다는 것을 실시간으로 검증할 수 있게하고, (2) 전달 시간을 최소로 하고, (3) 사용자의 사용패턴에 적응하도록 한다.
- 프로그램 분석을 통해서 미래 넷트웍 환경에 최적인 코드를 만드는 컴파일러 기술이 목표인 우리는, 프로그램 분석 실험의 일터를 실제 컴파일러 구축을 통해 실현할 필요가 있다. 기존의 ML 컴파일러들은 프로그램 분석 기술을 충분히 응용하고 있지 못하거나, 그 가능성을 염두에 두고 만들어지지 않았다.
- ML 컴파일러를 독자 개발할 필요가 있다. ML은 프로그래밍 이론의 성과(2세대 벅잡는 기술의 완성[4])를 담고 있으면서도, 실용적으로 고안된 언어이다. 이러한 언어의 컴파일러 기술은 미래 소프트웨어 산업의 근간이 된다.
- 기존의 명령형 언어(imperative language)의 전통을 고려한 ML 컴파일러가 필요하다. C, Java 등과 쉽게 접속(interoperation)할 수 있는 프로그래밍 시스템이 필요하다. 또한 명령형 언어에 익숙한 사람들이 쉽게 ML에 적응토록 하기 위해 기존과 유사한 환경을 제공하는 ML 컴파일러가 필요하다.

*본 연구는 과학기술부 창의적 연구진흥사업 추진으로 얻어진 결과임.
†E-mail: { kwang; cookcu; poisson; judaigi; wschoi; puppy; hgkang; saseo; abyss; bskim }@ropas.kaist.ac.kr

```
fun 실행 fp 프로그램 = case 프로그램 of
  App(_,함수이름,인자들) =>
    let val new_fp = !sp
        val 함수번호 = 함수찾기 fp 함수이름
        val (인자수,코드) = 함수표.[함수번호]
        val 값들 = Array.map (실행 fp) 인자들
        val s = Array.length 값들
    in  if 인자수 <> s then
          raise (Error "인자의 개수가 다릅니다")
        else
         (for i=0; i<s; i+1 do push 값들.[i] end;
          let val x = 실행 new_fp 코드
          in  sp := new_fp; x
          end)
    end
```

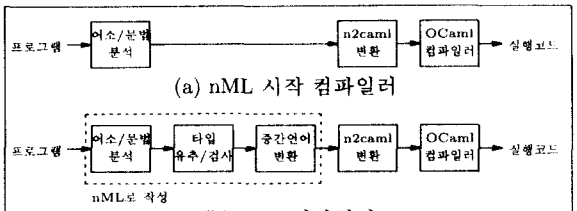
그림 1: nML로 작성된 예제 프로그램.

2 nML 언어

nML 언어는 SML[10]과 OCaml의 장점을 취합하여 엄밀하게 정의하였다. 기존의 명령형 언어에 익숙한 사용자들이 쉽게 적응할 수 있도록 명령형 특색이 반영되었다. nML 언어의 문법, 의미 구조, 타입 시스템은 [11]에 정의되어 있고, 예제 프로그램은 그림 1에 있다.

ML이 가지는 특징을 일별하면:

- 작고 간단하다.
- 정확한 의미구조를 갖추고 있어 엄밀한 분석이 가능하다.
- 첨단의 타입시스템 덕분에 프로그램 기획이 쉽다.
- 수행중인 프로그램의 타입 안전성(type safety)이 보장되어 있다.
- 정제된(simple and advanced) 모듈 처리기능을 갖추고 있어서 대형의 소프트웨어 개발이 용이하다.
- 자동으로 메모리를 관리해 준다.
- 위기상황 관리(exception handling) 매카니즘을 갖추고 있다.



(a) nML 시작 컴파일러

(b) nML 컴파일러

그림 2: nML 컴파일러의 내부 구조.

```
nML compiler 0.91 (built on 2001/5/2)
  Copyright(c) 2000-2001 Research On Program
  Analysis System, KAIST
  National Creative Research Initiative Center
  http://ropas.kaist.ac.kr/n
  Based on Objective Caml 3.01 Code

# fun fac(n) = if n=0 then 1 else fac(n=1);;
오류: int 타입이어야 하는데 bool 타입입니다.
```

그림 3: nML 컴파일러 실행화면과 타입 오류 메시지.

nML은 위의 전통적인 ML의 장점에 더해서, 그 컴파일러가 기존의 언어(C, Java등)과 소통(interoperability)할 수 있게하고, 정밀한 프로그램 분석기술을 통해서, 실시간 안전성 검증이 가능하고, 작고, 자발적인 적응력이 있는 코드를 만들어내도록 할 것이다.

3 nML 컴파일러

nML 컴파일러 0.91을 개발하였다. 두 가지 형태의 컴파일러를 제공하는데, 하나는 대화형으로 프로그램을 입력하면 바로 컴파일/실행하여 결과를 보여주는 `nml`이다. 다른 하나는 nML 프로그램을 화일로 작성해 주면 각각의 화일을 컴파일하여 링크시켜 실행가능한 코드로 만들어 주는 `nmlc`이다. `nmlc`는 gcc와 같은 기존의 전통을 유지한 형태이다. nML 컴파일러는 Unix, Linux, MS 윈도우즈 98/ME/NT/2000 에서 실행된다.

nML 컴파일러는 nML로 작성되었다. 이를 가능하게 하기 위해 우선 nML 시작 컴파일러를 개발하였다. nML 프로그램을 OCaml (Objective Caml) 프로그램으로 변환해 주는 변환기(n2caml 변환기)를 OCaml 컴파일러 앞에 장착하여 시작 컴파일러를 만들었다 (그림 2.(a)). 이렇게 만들어진 시작 컴파일러를 사용하여 nML로 작성된 컴파일러 소스을 컴파일하여 nML 컴파일러를 만들었다 (그림 2.(b)).

nML 컴파일러 0.91은 전단부가 독자적으로 개발되었다. 어소/문법 분석기 (lexer and parser), 타입 유추 및 검사기 (type checker), 중간 언어 변환기가 개발되었다. 남은 단계는 OCaml을 사용하여 구현되었지만, 추후에는 모두 nML 독자의 것으로 대체될 것이다.

- 어소/문법 분석 (lexer and parser):

  어소/문법 분석은 프로그램 텍스트(text)를 언어의 정의에 따라 트리 구조를 갖춘 AST(abstract syntax tree)로 변환해 주는 과정을 말한다.

  nML 정의를 따르는 명확한 문법 분석기(parser)를 구현하였다. 기존의 ML 컴파일러의 문법 분석기들은 여러 개의 충돌(conflict)이 있다. nML 문법 분석기는 단 한개의 충돌 없이 구현되었는데, 이는 문법 분석기가 명확하게 구현되었음을 뜻한다. 또한 nML 문법 분석기는 후처리가 필요하지 않아 AST를 이용한 프로그래밍 도구 작성에 용이하다.

  한글을 지원하여 한글이름을 사용하는 것이 가능하다 (그림 1). 그리고, 정보를 추가하기 쉬운 형태로 AST를 정의 하여, 분석 결과를 AST에 저장하기 용이하다.

- 타입 유추 및 검사 (type checker):

  타입 유추 및 검사기는 타입이 적혀 있지 않는 프로그램으로부터 타입을 유추해 내고, 잘못된 타입이 사용된 부분을 감지하여 발생 가능한 오류를 잡아 준다.

  타입 유추 및 검사기는 기존의 ML 컴파일러들이 사용한 타입 유추 알고리즘을 혼합한 알고리즘[8]으로 구현함으로써 오류 메시지의 이해도를 높였다. 또한 오류 메시지 역시 한글을 지원한다. 그림 3은 `nml`에서 타입이 맞지 않는 프로그램에 대한 오류 메시지의 예이다.

- 중간 언어 변환:

  타입 검사가 끝난 프로그램은 복잡한 구조를 단순한 구조로 변환하여 하위 수준의 언어로 변환한다. 구현된 중간 언어 변환 단계는 레코드(record) 컴파일과 패턴(pattern) 컴파일이다.

  레코드 컴파일은 레코드를 상대적으로 단순한 튜플(tuple)로 변환하는 과정이다. 레코드는 레이블(label)이 달린 값의 모음이다. 레이블이 없는 값의 모임인 튜플로 레코드에 관한 부분을 변환한다.

  패턴 컴파일은 nML의 복잡한 패턴을 단순한 패턴으로 변환하는 과정이다. nML의 복잡한 패턴은 하위 수준 언어에서 여러 번의 비교 과정을 통해 값이 패턴에 맞는지 결정된다. 한 패턴이 한 번의 비교로 결정될 수 있도록 단순한 패턴으로 변환한다.

- nML에서 OCaml로의 변환:

  0.91 컴파일러는 후단부로 OCaml 컴파일러를 사용하고 있다. 앞의 컴파일 과정을 거친 프로그램을 OCaml 프로그램으로 변환하여 OCaml 컴파일러를 통해 실행 가능한 코드를 얻는다.

nML 컴파일러는 테스트를 통해 신뢰성과 편리성을 확보하였다. 2 년간 프로그래밍 언어 수업의 숙제용 언어로 사용되어, 전체 217 명의 학생이 사용하였으며, 인터프리터, 타입 검사기 등이 구현되었다. 이를 통해 nML 컴파일러의 버그를 잡아냄으로써 신뢰성이 높아졌다. 사용자들의 불편함을 입력으로 nML을 보다 편리하게 사용하도록 개선하였다.

4 nML 프로그래밍 도구 및 응용 프로그램들

보다 쉽게 nML 프로그래밍을 할 수 있도록, 다음 프로그래밍 도구들을 제공한다. `nyacc`을 제외한 모든 도구들이 nML로 작성되었다.

- `nlex/nyacc`: 어소/문법 분석기 생성기.

  입력 텍스트(text)로부터 의미있는 토큰을 찾아 주는 어소 분석기와, 입력 토큰의 나열으로부터 문법에 따라 트리 형태의 구조로 만들어 주는 문법 분석기를 생성해 주는 프로그램이다.

- `nmlmake/nmldot`: nML 의존성 (dependency) 분석기.

  `nmlmake`는 nML 화일들 사이의 의존성을 분석하여, 의존 순서에 따라 차례대로 nML 화일들을 컴파일할 수 있도록 `Makefile`을 만들어 준다. `nmldot`는 의존성을 그래프로 그릴 수 있도록 `.dot` 화일[6]을 생성한다. lex 화일이나 yacc 화일의 경우 `nlex` 또는 `nyacc`을 호출하여 어소/문법 분석 프로그램을 생성한 후 의존성을 분석한다.

- `cfa`: nML 실행 흐름 분석기 (control-flow analyzer).

  프로그램 수행 중에 어떠한 흐름을 가지는지 예측해낸다. 분석 결과는 프로그램 안에 있는 각 함수 호출식에 대해서, 그 위치에서 호출될 가능성이 있는 함수들을 알려준다. 특히 여러 모듈로 이루어진 nML 프로그램은 전체 프로그램을 한 번에 분석할 필요없이 모듈 단위로 차례대로 분석할 수 있다.

- `ea`: nML 예외상황 분석기 (exception analyzer).

  각 함수식의 실행 중에 발생되지만 처리되지 않을 가능성이 있는 예외상황들을 예측해낸다. 분석 결과는 프로그램 안에 있는 각 함수에 대해서, 발생 가능한 예외상황의 이름과 만들어진 곳을 알려준다. 구현된 분석기는 Yi와 Ryu의 예외 상황 분석기[12]이다. SML/NJ 컴파일러의 경우[12]와 달리 모듈 단위 분석이 가능하다.

nML로 여러 응용 소프트웨어들을 구현함으로써 nML 언어의 유용성을 확인하였다. 또한 이런 구현 예들은 이후 다양한 소프트웨어 개발에 도움이 된다. 구현된 응용 소프트웨어들은 다음과 같다.

- XML/Java/C 문법 분석기.

  다른 언어에 대한 문법 분석기 구현은, 해당 언어들을 이용하는 응용 소프트웨어나 도구 개발에 필수적이다. 널리 쓰이는 XML/Java/C의 문법 분석기를 구현함으로써, nML로 작성할 수 있는 응용 소프트웨어의 범위가 넓어졌다.

- `netPhone`: 인터넷 전화 시스템 [3].

  이 프로그램의 구현은 nML이 복잡한 시스템 프로그래밍에도 사용할 수 있는 실제적인 언어임을 보여 준다. 일대일 통화 뿐만 아니라, 다자간 통화도 지원한다. 음성 데이타 처리, 사용자 인터페이스를 포함한 모든 부분이 nML로 구현되었다.

- `CloneChecker`: 프로그램 유사성 검사기 [1].

  프로그램들의 유사성을 비교해서, 비슷한 프로그램들을 묶어 보여준다. 프로그램 소스를 불법적으로 복사하여 사용하는 경우를 감지할 수 있다. 현재, C, Java, Scheme, nML 언어들에서 동작한다. 실제 강의에서 사용되어 많은 불법 복사된 숙제를 찾아 낼 수 있었다.

- `vmc`: 모델 체커 (model checker) 검증 시스템 [5].

  모델 체커를 다른 모델 체커로 검증하는 방법이 있는데, 문제는 모델 체커를 검증하는 체커를 믿을 수 있어야 한다. 이러한 믿을 수 있는 모델 체커 검증 시스템을 nML로 구현하였다.

- MySQL 라이브러리.

  공개 데이터 베이스 (database) 시스템인 MySQL을 nML에서 사용할 수 있도록 하였다. C로 구현된 MySQL 라이브러리를 nML에서 연동할 수 있도록 하였다.

- SOAP[7] 라이브러리.

  XML을 이용하여 원격 함수 호출(remote procedure call)을 가능케한 프로토콜 SOAP을 nML에서 사용할 수 있도록 하였다. SOAP 방식의 컴포넌트들을 자유롭게 nML에서 활용할 수 있다.

5 결론

상위의 타입 구조를 갖춘 nML 언어를 고안하고 그 컴파일러를 개발하였다. nML은 프로그래밍 언어 분야의 중요한 성과들을 충실히 실현한 언어로서, 실제 프로그래밍 세계의 관성을 염두에 두고 정의되었다. nML 컴파일러의 실용성과 안전성은 이미 여러 응용 시스템 소프트웨어의 개발로 실험되고 있다.

모든 nML에 대한 자료는

`http://ropas.kaist.ac.kr/n`

를 통해 공개 되고 있다.

참고 자료

[1] 장성순, 서선애, 이광근. 프로그램 유사성 분석기. *한국정보과학회 추계 학술회의* 논문집, 2001년 10월.

[2] Standard ML of New Jersey, version 110, 1998. `http://cm.bell-labs.com/cm/cs/what/smlnj`.

[3] 김범식. nML을 이용한 인터넷 전화 시스템의 구현과 검증. 석사논문, 한국과학기술원 전자전산학과, 2001.

[4] 이광근. nML programming system for safe global computing. 한국과학기술원 전산학전공 학과 콜로키엄, May 2001. `http://ropas.kaist.ac.kr/~kwang`.

[5] 어현준, Nikolay V. Shilov, 이광근. 모델체커를 검증하기 위한 모델체커. *한국정보과학회 추계 학술회의* 논문집, 2001년 10월.

[6] Graphviz: a graph drawing software. AT&T Research. `http://www.research.att.com/sw/tools/graphviz`.

[7] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocal (SOAP) 1.1, May 2000. `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`.

[8] Oukseh Lee and Kwangkeun Yi. A generalized let-polymorphic type inference algorithm. Technical Memorandum ROPAS-2000-5, Research on Program Analysis System, KAIST, March 2000. `http://ropas.kaist.ac.kr/memo`.

[9] Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The Objective Caml system (release 3.01), documentation and user's manual, 2001. `http://caml.inria.fr/ocaml/htmlman/index.html`.

[10] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[11] *The Definition of nML*. Research On Program Analysis System, KAIST, July 2001. `http://ropas.kaist.ac.kr/n/doc/n.ps, n.pdf`.

[12] Kwangkeun Yi and Sukyoung Ryu. Towards a cost-effective estimation of uncaught exceptions in SML programs. In *Proceedings of the 4th International Static Analysis Symposium*, volume 1302 of *Lecture Notes in Computer Science*, pages 98-113. Springer-Verlag, 1997.