

요리용 코딩언어 개발

-요리 레시피 기술을 위한 도메인 특화 언어 Swifood-



정보컴퓨터공학부 202055570 윤선재

정보컴퓨터공학부 201824598 채문석

정보컴퓨터공학부 202055619 팜민두웅

지도교수 조환규

목 차

1. 서론	1
1.1. 연구 배경	1
1.2. 기존 문제점	1
1.3. 관련 연구	3
1.4. 연구 목표	4
2. 연구 배경	6
2.1. 배경 지식	6
2.2. 사용 기술	6
2.3. 개발 환경	7
3. 연구 내용	8
3.1. 언어 설계	8
3.1.1. 문법 규칙	8
3.1.2. 타입 시스템	12
3.1.3. 명명 규칙	14
3.1.4. 언어 예시	15
3.2. 시스템 아키텍처	17
3.2.1. 전체 시스템 구성도	17
3.2.2. 데이터 흐름	18
3.3. 핵심 모듈 구현	25
3.3.1. 데이터 관리자 구현	25
3.3.2. 레시피 오류 분석 구현	26

3.3.3.	필요 도구 자동 분석 구현.....	27
4.	연구 결과 분석 및 평가.....	28
4.1.	기능별 결과	28
4.1.1.	언어 파싱 및 해석.....	28
4.1.2.	오류 검출 및 처리.....	29
4.1.3.	필요 도구 자동 분석	31
5.	결론 및 향후 연구 방향	32
6.	참고 문헌.....	33

1. 서론

1.1. 연구 배경

요리 레시피는 본디 인간이 이해하기 쉬운 자연어로 작성되어 이를 기계가 직접 해석하여 조리 동작을 수행하기에는 많은 모호성과 생략된 정보를 내포한다. 이는 최근 인공지능 및 컴퓨터 비전 기술의 발전으로 널리 개발되고 있는 로봇을 이용한 자동화 조리에 약점으로 작용한다. 이 문제를 해결하기 위해 지난 20년간 도메인 특화 언어(DSL)로 레시피를 형식화하여 로봇이 요리를 정확하게 반복적으로 수행할 수 있도록 활발히 연구되고 있다. 본 보고서는 이러한 최신 연구 및 상용 사례를 중심으로 요리 레시피 코드화의 동향을 살펴보고, 기존 기술의 문제점을 해결하는 방법을 제시하고자 한다.

1.2. 기존 문제점

요리 과정을 컴퓨터가 이해할 수 있는 도메인 특화 언어(DSL)로 표현하려는 시도는 여러 차례 있었다. 이러한 DSL은 레시피의 재료, 조리 단계, 조리 기구 등을 형식화하여 알고리즘처럼 기술하고자 한다. 기존에 시도된 구현 사례의 특징 및 한계는 다음과 같다.

1.2.1 레시피ML (Recipe Markup Language)

2000년경 제안된 XML 기반 포맷으로 코드명 DESSERT(Document Encoding and Structuring Specification for Electronic Recipe Transfer)로도 불린다. XML 태그를 통해 레시피의 재료 목록과 단계별 지시사항을 마크업 함으로써 서로 다른 단위 환산을 자동화하거나 재료를 체계적으로 표현하려는 목적이 있었다[1]. 하지만 RecipeML은 산업 표준으로 널리 채택되지는 못했고, 다양한 레시피 포맷이 혼재하는 등의 한계가 있다.

1.2.2 Corel (Cooking Recipe Language)

2021년 네덜란드 흐로닝언대(University of Groningen)의 연구로 개발된 요리 레시피 DSL이다. Rascal 언어 워크벤치를 이용하여 구현되었으며, 레시피의 구조를 도메인 분석(FODA 기법)으로 모델링하여 설계되었다. Corel은 재료에 대

한 이해와 영양 성분 계산 등의 기능을 포함하고 있어 레시피로부터 영양 라벨을 자동 생성할 수 있다. 실제 요리 동작의 모든 측면을 기술하기에는 한계가 있지만 재료, 분량, 단계 등을 형식화하여 레시피 지식을 컴퓨터가 처리할 수 있음을 보였다는 의의가 있다[2].

1.2.3 Cooklang

2020년대에 커뮤니티 주도로 만들어진 오픈소스 레시피 마크업 언어이다. 일반 텍스트로 작성된 레시피에 특수 문자를 넣어 재료, 용기, 조리 시간 등의 정보를 기계가 파싱할 수 있게 하였다. 예를 들어 2킬로그램의 감자는 '@감자 {2kg}'의 형태로 표현한다. 또한 각 줄을 단계로 구분하는 등 단순한 문법을 제공하며[3], 개발 도구와 모바일 앱 등의 생태계를 갖추어 개인 레시피 관리나 장보기 리스트 연계 등에 활용되고 있지만, 이는 주로 데이터 관리 목적이며 로봇 제어를 위한 완전한 절차 명세로 쓰이기에는 표현력에 한계가 있다.

항목	RecipeML	Corel	Cooklang
제안 시기	2000년경	2021년	2020년대
표현 방식	XML 마크업	Rascal 기반 DSL	특수 기호가 포함된 일반 텍스트
구현 목적	단위 변환, 재료 체계화	레시피 지식 구조화, 영양 계산	레시피 관리, 앱 연동
표현력	중간 (재료, 단계 위주)	높음(재료, 영양소, 조리 단계 등)	낮음(기본 정보 중심)
기계 해석 가능성	있음(XML 기반 구조화)	높음(도메인 모델링 기반)	제한적(단순한 문법 구조)
활용 사례	제한적(표준화 실패)	연구용 도구로 사용됨	모바일 앱, 도구 등 존재
한계점	보급 부족, 포맷 난립	실제 조리 절차 표현은 미흡	표현력 부족, 로봇 제어에는 부적합

표 1. 요리 레시피 DSL 목록

1.3. 관련 연구

최근 비약적으로 발전 중인 인공지능 및 컴퓨터 비전 기술은 여러 분야들에 접목되어 활발하게 활용되고 있다. 이 중 요리와 인공지능 및 컴퓨터 비전을 융합하여 개발 중인 자동 조리 로봇은 현재 상용 시스템과 연구 프로토타입 양쪽 모두에서 비약적으로 발전 중이며, 이러한 기술에는 다음과 같은 것들이 있다.

1.3.1 BakeBot

2010년대 초 MIT에서 진행한 프로젝트로 PR2 로봇을 이용하여 쿠키 반죽 만들기 등 베이킹 작업을 수행하며 자연어 레시피를 로봇 명령으로 해석하는 실험을 하였다[4]

1.3.2 Moley Robotic Kitchen

2017년대에 개발된 대표적 사례로 주방 전체를 하나의 로봇 셰프로 구성하여 로봇 팔이 조리 동작을 수행하도록 설계됐다. 셰프의 움직임을 그대로 모방하는 것을 목표로 하였으며, 구획된 공간에서 조리 후 자체 세척까지 수행하는 컨셉을 제시했다. 하지만 복잡한 기술과 높은 비용 때문에 일반 가정에 보급되지는 않았다[5].

1.3.3 Bot Chef

삼성 등 대기업이 CES 2020 등에서 공개한 사례로 두 개의 로봇 팔이 조리대에서 작업을 수행하고 인간과 협동하며 보조적인 역할을 할 수 있는 주방 로봇 팔로 주목받았으나 실제 상용 제품으로 출시되지는 않았다[6].

1.3.4 YORI 시스템

UCLA 연구팀 등이 제안한 모듈형 자동 주방 시스템으로 협동 로봇(cobot)을 이용하여 조리대, 도구 배치 등을 유연하게 변경하며 다양한 요리를 구현하는 연구가 진행됐다[7].




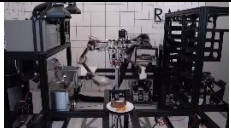
항목	BakeBot	Moley Robotic Kitchen	Bot Chef	YORI 시스템
사진	 [8]	 [9]	 [10]	 [11]
개발 시기 및 기관	2010년대 초 MIT	2017년경 Moley Robotics	2020년경 삼성 등	2020년대 UCLA 등
조리 범위	베이킹 중심(반죽 등)	전체요리(셰프 동작 모방)	조리 보조 중심	조리 보조 중심
로봇 유형	PR2 이족형 로봇	전용 로봇 팔 2개 + 주방 통합 시스템	로봇 팔 2개(협동형)	협동 로봇 (cobot) 기반
레시피 해석 방식	자연어 레시피 → 명령 해석	사전 녹화된 셰프 동작 모방	명령 기반 조작, 비전 활용 가능	DSL 등 연구 가능성 있음
자동화 수준	반자동	완전 자동	반자동, 인간 협동형	유연한 자동화 구성 가능
상용화 여부	연구용	상용화 시도(비싼 가격)	전시용 시연, 미출시	연구용 프로토타입
기술적 한계	인식 정확도, 기능 제한	고비용, 설치 공간 제한	제품화 미진, 실용성 부족	구성 복잡성, 시스템 통합 문제

표 2. 자동 조리 시스템 목록

1.4. 연구 목표

본 과제는 조리 동작의 표현을 중점으로 한 요리 레시피 DSL을 설계하고, 이를 실제로 사용할 수 있는 환경을 구현하는 것을 목표로 한다. 이를 통해 요리 레시피 기술의 새로운 방법을 제시함으로써 다음과 같은 성과를 지향한다.

1.4.1 요리 과정의 형식화

1.4.1.1 조리 단계, 재료, 도구, 시간 등 레시피를 구성하는 요소들을 명확하게

정의할 수 있도록 한다.

1.4.1.2 재료 표현, 조리 시간, 종료 조건 등에서 사용되는 단위를 SI 단위계 기반으로 표준화한다.

1.4.2 요리 과정의 일관된 해석 구조 구현

1.4.2.1 재료 용량이나 조리 시간 등을 표준 단위로 표현하여 지역이나 문화 등에 따른 해석의 차이를 방지한다.

1.4.3 인간 친화적 출력 기능 제공

1.4.3.1 코드 형태의 레시피를 사람이 쉽게 이해할 수 있는 자연어 형식으로 출력할 수 있는 기능을 포함한다.

1.4.4 레시피 작성을 위한 편의 기능 제공

1.4.4.1 레시피 오류 검사 기능을 통해 작성한 레시피 코드의 오류를 쉽게 파악하고 수정할 수 있도록 한다.

1.4.4.2 필요한 도구 자동 분석 기능을 통해 작성된 레시피에서 필요한 도구를 자동으로 파악할 수 있도록 한다.

2. 연구 배경

2.1. 배경 지식

2.1.1 도메인 특화 언어(Domain Specific Language, DSL)

도메인 특화 언어는 특정한 애플리케이션 도메인에 특화된 언어로 범용 언어(General-Purpose Language, GPL)에 반대되는 개념이다. DSL의 대표적인 예시로 웹페이지에 특화된 HTML과 데이터베이스에 특화된 SQL 등이 있다[12].

DSL은 특정 용도에 특화되어 범위가 제한적이라는 특징을 가진다. 이러한 특성 덕분에 여러 장점을 제공한다. 첫째, 학습하기 용이하여 비교적 쉽게 사용할 수 있다. 둘째, 특정 도메인에서 사용되는 것을 가정하므로 도메인 수준에서 검증을 허용하여 높은 안정성과 신뢰성을 확보할 수 있다.

2.2. 사용 기술

2.1.1 GCC / C++

GCC(GNU Compiler Collection)은 GNU 프로젝트에 의해 만들어진 오픈소스 컴파일러 모음으로 C, C++, Objective-C, Fortran, Ada 등 다양한 프로그래밍 언어를 지원한다[13].

C++는 비야네 스트루스트룹(Bjarne Stroustrup)에 의해 설계된 프로그래밍 언어로 C언어에 객체지향 프로그래밍(Object-Oriented Programming) 요소를 추가한 확장판으로 시작하였다[14]. C++는 시스템 프로그래밍과 애플리케이션 프로그래밍을 비롯한 여러 분야에서 사용되고 있으며, 현재는 함수형 프로그래밍(Functional Programming), 제네릭 프로그래밍(Generic Programming) 등 다양한 요소들을 추가하여 다중 패러다임(Multi-Paradigm) 언어가 됐다[15].

2.1.2 Flex

Flex(Fast lexical analyzer generator)는 번 팩슨(Vern Paxson)이 개발한 어휘 분석기 생성기(Lexical analyzer generator)이다[16]. 어휘 분석기(Lexical Analyzer)는 텍스트를 분석하여 사전에 정의된 규칙에 따라 식별자, 연산자, 상수 등 의미를 가지는 최소 단위인 토큰(Token)으로 분해하는 역할을 하며 스캐너(Scanner) 또는 렉서(Lexer)라고도 불린다.

2.1.3 Bison

Bison은 GNU 프로젝트의 일부인 파서 생성기(Parser generator)이다[17]. 파서는 구문 분석(Syntax Analysis)를 담당하며 lex나 Flex 등의 어휘 분석기가 생성한 토큰들이 정의된 규칙에 맞게 올바른 순서로 배열되었는지 검사하며, 만약 문법에 어긋날 경우 오류를 보고하고, 올바른 경우 추상 구문 트리(Abstract Syntax Tree)와 같은 자료구조를 생성하여 다음 컴파일 단계로 넘겨준다.

2.1.4 Make

Make는 스튜어트 펠드먼(Stuart Feldman)이 개발한 소프트웨어 빌드 도구로 프로젝트의 빌드 과정을 자동화하는데 사용된다[18]. Makefile에 소스 코드 파일 간의 의존성과 빌드 규칙을 미리 정해두면 make 명령어 하나로 컴파일, 링크 등 복잡한 빌드 과정을 자동으로 처리해준다.

2.3. 개발 환경

2.2.1 macOS Sequoia 15.6.1

2.2.2 Visual Studio Code

2.2.3 zsh

3. 연구 내용

3.1. 언어 설계

3.1.1. 문법 규칙

Swifood의 코드는 크게 Ingredient 블록, Action 블록, 그리고 Recipe 블록, 총 3가지 블록으로 나뉜다. 각 블록은 해당하는 요소의 정의를 의미하며, 각 요소별 특징은 다음과 같다.

3.1.1.1 Ingredient

Swifood에서 Ingredient는 요리 레시피의 재료를 의미한다. Ingredient 블록은 "Ingredient <CategoryName> { /* Ingredients */}"의 형태를 가지며, 현재 라이브러리에서 사용 가능한 재료의 종류를 정의하는 역할을 담당한다. Ingredient 블록의 카테고리 지정은 다음과 같이 두 가지의 형태를 가진다.

```
Ingredient {  
    Water,  
    Salt,  
    Egg  
}
```

그림 1. 기본 카테고리

```
Ingredient Vegetable {  
    Lettuce,  
    Carrot  
}
```

그림 2. 야채 카테고리

기본 카테고리는 별도의 카테고리가 지정되지 않은 재료를 의미하며, 이러한 카테고리들은 레시피에 사용된 재료들의 카테고리를 파악하기 위한 것으로 향후 추가될 기능을 위해 포함됐다.

3.1.1.2 Action

Swifood에서 Action은 요리 레시피의 동작을 의미한다. Action 블록은 "Action <ActionName> <Target-Type> { /* Action-Parameters */ }"의 형태를 가지며, 현재 라이브러리에서 사용 가능한 조리 동작을 정의하는 역할을 담당한다. Action Parameter는 "<Parameter-Keyword>: <ParameterValue>"의 형태를 가지며, Parameter-Keyword에는 in, with, on, to, for, until 등이 존재하고, Parameter Value에는 Ingredient type, Equipment type, List type, Target-Type type 등의 값이 사용될 수 있다. 실제 액션의 정의에서 나타나는 파라미터의 예시는 다음과 같다.

```
Action mix $I {  
  in: bowl,  
  with: spoon  
}
```

그림 3. 기본 파라미터만 존재하는 액션 mix

```
Action preheat $E {  
  to: $T  
}
```

그림 4. 요구 파라미터만 존재하는 액션 preheat

```
Action boil $I {  
  in: pot,  
  (for: $D | until: $C)  
}
```

그림 5. 기본 파라미터 in과 XOR 파라미터 (for|until)이 존재하는 액션 boil

```
Action wash $I {}
```

그림 6. 파라미터가 존재하지 않는 액션 wash

액션의 파라미터에는 기본 파라미터(Default Parameter), 요구 파라미터(Required Parameter), 그리고 XOR 파라미터(XOR Parameter)가 존재한다. 기본 파라미터는 파라미터 값으로 Ingredient type, Equipment type, Duration type, Condition type 등의 값을 가지는 파라미터로 파라미터 값이 기본적으로 정해져 있으며, 액션 호출시에 파라미터 값을 별도로 지정하지 않으면 기본값이 사용됨을 의미한다. 요구 파라미터는 Target-Type type의 값을 가지는 파라미터로 액션 호출시에 파라미터 값으로 지정된 Target-Type에 해당하는 값을 반드시 지정해야 함을 의미한다. XOR 파라미터는 향후 추가될 파라미터의 종류로 지정된 파라미터들 중 하나의 파라미터 만을 지정해야 함을 의미한다. 액션의 호출시에 파라미터는 액션 정의에서 정의된 파라미터들만 지정할 수 있으며, 어떠한 파라미터도 가지지 않은 액션 wash의 경우 액션 호출시에 빈 파라미터를 전달해야 함을 의미한다.

3.1.1.3 Recipe

Swifood에서 Recipe는 요리 레시피를 의미한다. Recipe 블록은 "Recipe <RecipeName> { Ingredients: /* IngredientDeclarations */ Steps: /* Steps */ }"의 형태를 가지며, 현재 레시피에서 사용 가능한 레시피를 정의하는 역할을 담당한다. 재료 선언은 "<IngredientName> <Amount>"과 같은 형태를 가지며, 레시피에서 사용할 재료를 지정하는 역할을 담당한다. 스텝은 C언어의 함수 호출과 유사하며, "<ActionName> <Target> (/* parameters */)."의 형태를 가지고, 레시피가 수행할 동작들을 정의하는 역할을 담당한다. 실제 레시피의 정의는 다음과 같다.

```
Recipe Boiled_Egg {
  Ingredients:
    Egg 4ea, Salt 8g, Water.

  Steps:
    put [Water, Salt] (in: pot).
    heat pot (until: boils).
    boil Egg (for: 00:07:00).
}
```

그림 7. 레시피 정의

Swifood에서 레시피는 재료의 일종으로 취급되며, 이는 레시피에서 또 다른 레시피의 호출이 가능함을 의미한다.

```
Recipe Boiled_Egg {
  Ingredients:
    Egg 4ea, Salt 8g, Water.

  Steps:
    put [Water, Salt] (in: pot).
    heat pot (until: boils).
    boil Egg (for: 00:07:00).
}

Recipe Double_Boiled_Egg {
  Ingredients:
    Boiled_Egg 4ea, Water.

  Steps:
    put [Boiled_Egg, Water] (in: pot).
    heat pot (until: boils).
    boil Boiled_Egg (for: 00:07:00).
}
```

그림 8. 레시피에서 레시피 호출 코드

```

[sunjae@Sunjae-MacBook-Air swifood % ./swifood Double_Boiled_Egg
[Double_Boiled_Egg]

--- Equipments ---
    pot

--- Ingredients ---
    * Boiled_Egg 4ea
    * Water

--- Method ---
    1. Put Boiled_Egg, and Water in pot.
    2. Heat pot until boils.
    3. Boil Boiled_Egg in pot for 00:07:00.

--- Related recipes ---
[Double_Egg]

--- Equipments ---
    pot

--- Ingredients ---
    * Egg 4ea
    * Salt 8g
    * Water

--- Method ---
    1. Put Water, and Salt in pot.
    2. Heat pot until boils.
    3. Boil Egg in pot for 00:07:00.

```

그림 9. 레시피에서 레시피 호출 결과

3.1.2. 타입 시스템

Swifood에는 다양한 타입들이 존재하며, 각 타입들은 정해진 블록에서만 사용할 수 있다. 각 블록별로 사용 가능한 타입들은 다음과 같다.

3.1.2.1 공용

3.1.2.1.1 List type

C의 배열에 해당하는 타입으로 여러 요소들을 지정하는 역할을 하며 “[<Element-1>, <Element-2>, ... <Element-n>]”의 형태를 가진다. Action 블록의 기본 파라미터와 Recipe 블록의 스텝에서 타겟이나

파라미터 값에 사용된다.

3.1.2.1.2 Ingredient type

레시피의 재료를 나타내며 Ingredient 블록에선 새로운 Ingredient 를 정의할 수 있고, Action 블록의 기본 파라미터와 Recipe 블록의 스텝에서 타겟이나 파라미터 값에 사용되며, Recipe 블록의 재료 선언에도 사용된다.

3.1.2.1.3 Equipment type

레시피에서 사용되는 도구를 나타내며 Action 블록의 기본 파라미터와 Recipe 블록의 스텝에서 타겟이나 파라미터 값에 사용된다.

3.1.2.1.4 Duration type

액션의 파라미터 중 시간을 나타내는 타입으로 Action 블록의 for 기본 파라미터 값과 Recipe 블록의 스텝에서 for 파라미터 값에 사용된다.

3.1.2.1.5 Condition type

액션의 파라미터 중 종료 조건을 나타내는 타입으로 Action 블록의 until 기본 파라미터 값과 Recipe 블록의 스텝에서 until 파라미터 값에 사용된다.

3.1.2.1.6 Temperature type

액션의 파라미터 중 온도를 나타내는 타입으로 Action 블록의 to 기본 파라미터 값과 Recipe 블록의 스텝에서 to 파라미터 값에 사용된다.

3.1.2.1.7 Action type

C의 함수에 해당하는 타입으로 조리 동작을 나타내며, Action 블록을 통해 새로운 액션을 정의할 수 있고, Recipe 블록의 스텝에선 레

시피에서 사용되는 액션을 지정할 수 있다.

3.1.2.2 Action 블록

3.1.2.2.1 Target-Type type

Action 블록의 타겟 지정에서 사용되는 타입으로 타겟의 타입을 지정하는 역할을 한다. 타입의 값으로 재료를 지정하는 "\$I"와 도구를 지정하는 "\$E"가 있다.

3.1.2.2.2 Parameter-Type type

Action 블록의 요구 파라미터에서 사용되는 타입으로 파라미터의 타입을 지정하는 역할을 한다. 타입의 값으로 시간을 지정하는 "\$D", 종료 조건을 지정하는 "\$C", 온도를 지정하는 "\$T" 등이 있다.

3.1.2.3 Recipe 블록

3.1.2.3.1 Ingredients 블록

3.1.2.3.1.1 Amount type

레시피에서 사용되는 재료의 용량 명시에 필요한 타입으로 재료 선언에서 "<IngredientName> <Amount>"의 형태로 사용된다.

3.1.3. 명명 규칙

Swifood는 코드의 가독성을 위하여 식별자의 이름을 엄격히 제한한다. 각 타입별 식별자의 명명 규칙은 다음과 같다.

3.1.3.1 Ingredient type

대문자로 시작되는 영단어와 "_" 구분자의 조합, 숫자 사용 불가, 식별자의 시작은 반드시 대문자, "_" 구분자는 영단어의 사이에만 사용 가능.

예) Ingredient_Name, Ing_Name

3.1.3.2 Action type

소문자로 이루어진 영단어와 "_" 구분자의 조합, 숫자 사용 불가, "_" 구분자는 영단어의 사이에만 사용 가능.

예) action_name, act_n

3.1.3.3 Recipe type

Ingredient type과 동일.

3.1.3.4 Category type

Ingredient type과 동일.

3.1.3.5 Equipment type

Action type과 동일.

3.1.3.6 Duration type

hh:mm:ss의 형태로 이루어진 시간 표기법, hh, mm, ss는 숫자로 표기.

예) 00:07:00, 01:10:30

3.1.3.7 Temperature type

숫자로 이루어진 수치와 단위 표기법의 조합, 단위 표기법은 C(Celsius), F(Fahrenheit), 그리고 K(Kelvin)을 사용.

예) 100C, 523F, 373K

3.1.4. 언어 예시

Swifood의 실제 코드 예시는 다음과 같다.

```
Ingredient {
  Water,
  Salt,
  Egg
}

Ingredient Vegetable {
  Lettuce,
  Carrot
}
```

그림 10. 재료 정의 예시

```
Action boil $I {
  in: pot,
  (for: $D | until: $C)
}

Action fry $I {
  in: pan,
  (for: $D | until: $C)
}

Action mix $I {
  in: bowl,
  with: spoon
}
```

그림 11. 액션 정의 예시

```
Recipe Boiled_Egg {
  Ingredients:
    Egg 4ea, Salt 8g, Water.

  Steps:
    put [Water, Salt] (in: pot).
    heat pot (until: boils).
    boil Egg (for: 00:07:00).
}

Recipe Test_Analyze_Equipments {
  Ingredients:
    Egg 4ea.

  Steps:
    boil Egg (for: 00:07:00).
}
```

그림 12. 레시피 정의 예시

3.2. 시스템 아키텍처

3.2.1. 전체 시스템 구성도

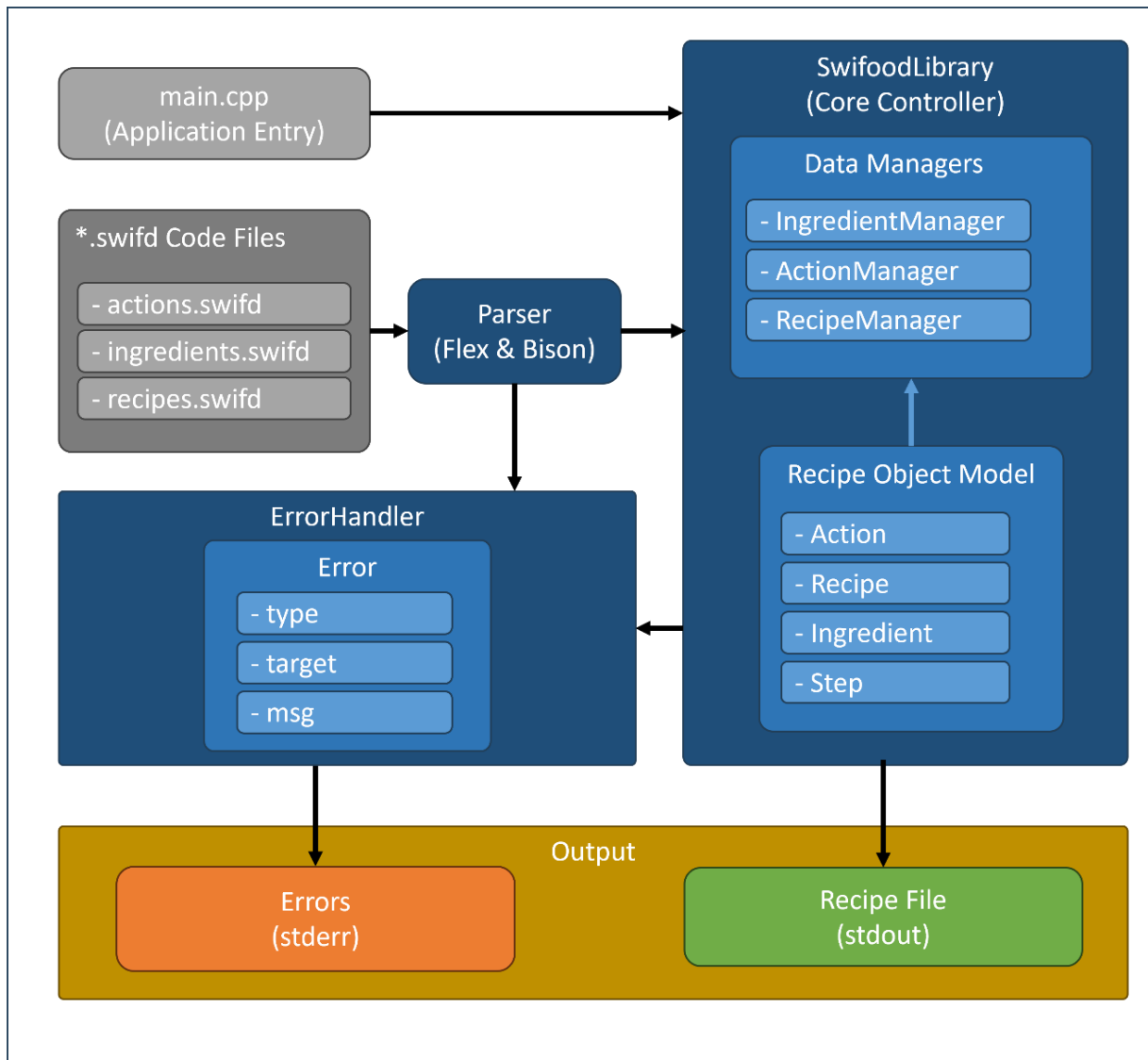


그림 13. 전체 시스템 구성도

3.2.2. 데이터 흐름

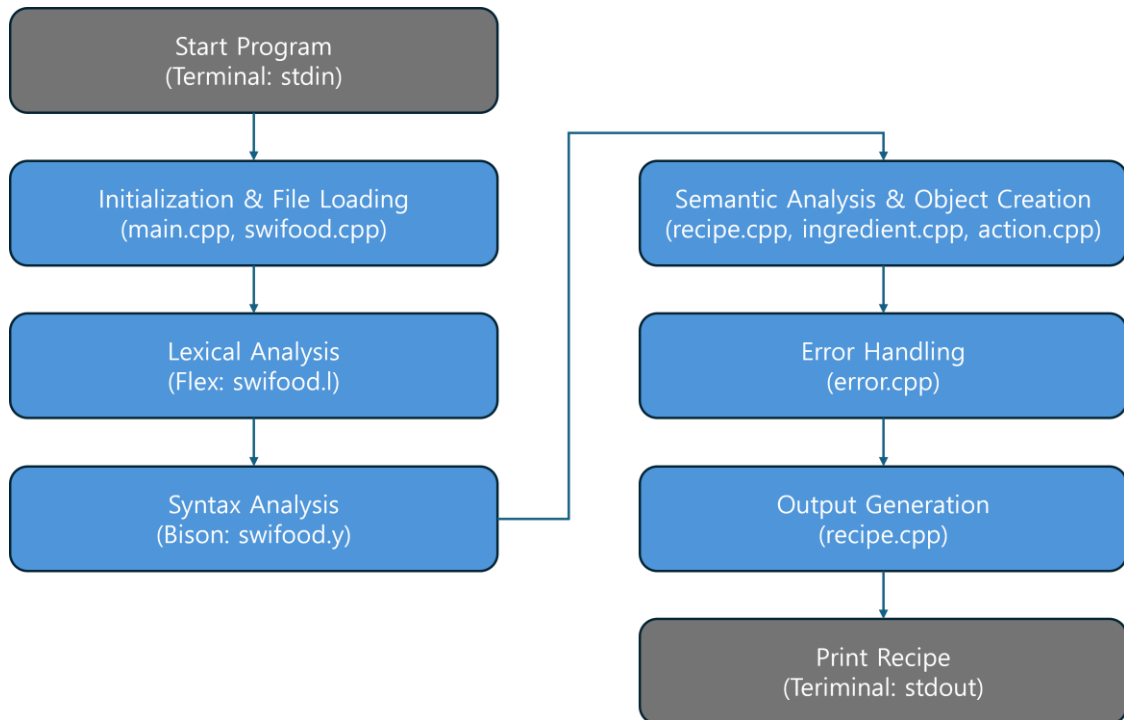


그림 14. 데이터 흐름도

3.2.2.1 초기화 및 파일 로딩(Initialization & File Loading)

```
int main(int argc, char* argv[])
{
    g_swifood = new SwifoodLibrary();

    if (argc < 2) {
        std::cerr << "Usage: " << argv[0] << " <recipeName>" << std::endl;
        return 1;
    }

    g_swifood->load(fileName: "actions.swifd");
    g_swifood->load(fileName: "ingredients.swifd");
    g_swifood->load(fileName: "recipes.swifd");

    Recipe recipe(definition: g_swifood->recipeManager.get_definition(name: argv[1]));

    if (ErrorHandler::get_instance().is_error_exist()) {
        ErrorHandler::get_instance().print_errors();
    } else {
        recipe.print();
    }

    delete g_swifood;

    return 0;
}
```

그림 15. main.cpp의 main 함수

사용자는 터미널에 `./swifood <RecipeName>`와 같은 명령어를 입력하여 프로그램을 시작한다. 프로그램이 시작되면 `main.cpp`의 `main` 함수에서 `SwifoodLibrary`의 전역 인스턴스 `g_swifood`가 생성되고 `g_swifood`의 `load()` 함수를 호출하여 `swifood` 파일 `actions.swifd`, `ingredients.swifd`, `recipes.swifd`를 순차적으로 읽어들인다.

3.2.2.2 파싱 1: 어휘 분석(Lexical Analysis)

```
void SwifoodLibrary::load(const std::string& fileName)
{
    FILE* inputFile = fopen(filename: fileName.c_str(), mode: "r");

    if (inputFile == nullptr) {
        std::cerr << "Error: Cannot open file " << fileName << std::endl;
        return;
    }

    yylineno = 1;
    yyrestart(new_file: inputFile);
    g_currentFileName = fileName;

    yyparse();

    fclose(inputFile);
}
```

그림 16. `swifood.cpp` 파일의 `SwifoodLibrary::load` 함수

`load` 함수는 표준 입출력 라이브러리와 `Flex`에서 생성된 함수들을 사용하여 파일을 불러온 후 파싱을 수행한다.

DIGIT	[0-9]
NUMBER	[0-9]+
DURATION	[0-9]{2}":"[0-9]{2}":"[0-9]{2}
TEMPERATURE	{NUMBER}[CFK]
AMOUNT	{NUMBER}[a-zA-Z]*
LOWERCASE_WORD	[a-z]+
CAPITALIZED_WORD	[A-Z][a-z]*
LOWERCASE_ID	{LOWERCASE_WORD}(_{LOWERCASE_WORD})*
CAPITALIZED_ID	{CAPITALIZED_WORD}(_{CAPITALIZED_WORD})*

그림 17. 정규표현식 이름 정의

```

%%
"Action"      { return T_DEF_ACTION; }
"Ingredient"  { return T_DEF_INGREDIENT; }
"Recipe"      { return T_DEF_RECIPE; }

"{"           { return T_LBRACE; }
"}"           { return T_RBRACE; }
"["           { return T_LBRACKET; }
"]"           { return T_RBRACKET; }
"("           { return T_LPAREN; }
")"           { return T_RPAREN; }
":"           { return T_COLON; }
","           { return T_COMMA; }
"."           { return T_DOT; }

"$I"          { return T_PARAM_INGREDIENT; }
"$E"          { return T_PARAM_EQUIPMENT; }
"$D"          { return T_PARAM_DURATION; }
"$C"          { return T_PARAM_CONDITION; }
"$T"          { return T_PARAM_TEMPERATURE; }

"in"          { return T_PARAM_IN; }
"with"        { return T_PARAM_WITH; }
"on"          { return T_PARAM_ON; }
"to"          { return T_PARAM_TO; }
"for"         { return T_PARAM_FOR; }
"until"       { return T_PARAM_UNTIL; }

"Ingredients" { return T_RECIPE_INGREDIENTS; }
"Steps"       { return T_RECIPE_STEPS; }

{LOWERCASE_ID}    { yyval.sval = new std::string(yytext); return T_ID_LOWERCASE; }
{CAPITALIZED_ID} { yyval.sval = new std::string(yytext); return T_ID_CAPITALIZED; }

{DURATION}       { yyval.sval = new std::string(yytext); return T_DURATION; }
{TEMPERATURE}    { yyval.sval = new std::string(yytext); return T_TEMPERATURE; }
{AMOUNT}         { yyval.sval = new std::string(yytext); return T_AMOUNT; }

[ \t]+          ; /* Skip whitespace */
[\n]            { yylineno++; }
.               { std::cerr << "Lexer Error: Unexpected character " << *yytext << std::endl; }
%%

```

그림 18. 정규표현식 규칙 정의

Flex는 .swifd 파일을 전달받아 정의된 규칙에 따라 토큰들로 분해하는 어휘 분석을 수행한다. 예를 들어 "Recipe Recipe_Name { Ingredients: /* Ingredients */ Steps: /* Steps */ }"는 규칙에 따라 "T_DEF_RECIPE T_ID_CAPITALIZED T_LBRACE T_RECIPE_INGREDIENTS T_COLON /* Ingredients */ T_RECIPE_STEPS T_COLON /* Steps */ T_RBRACE"와 같이 분해된다.

3.2.2.3 파싱 2: 구문 분석(Syntax Analysis)

```
program:
| definitions
;

definitions:
/* empty */
| definitions definition
;

definition:
ingredient_definition
| recipe_definition {
    g_swifood->recipeManager.add_recipe(*$1);
    delete $1;
}
| action_definition {
    g_swifood->actionManager.add_action(*$1);
    delete $1;
}
;

ingredient_definition:
T_DEF_INGREDIENT T_ID_CAPITALIZED T_LBRACE capitalized_identifiers T_RBRACE
{
    g_swifood->ingredientManager.add_ingredients(*$2, *$4);
    delete $2; delete $4;
}
| T_DEF_INGREDIENT T_LBRACE capitalized_identifiers T_RBRACE
{
    g_swifood->ingredientManager.add_ingredients("", *$3);
    delete $3;
}
;
```

그림 19. swifood.y 파일 중 일부

Flex에서 생성된 토큰 스트림은 Bison으로 전달되어 구문 분석을 수행한다. Bison은 정의된 문법에 따라 토큰들이 올바른 순서와 구조로 배치되었는지 분석하고, 문법 규칙이 일치한 경우 연결된 C++ 코드를 실행하여 데이터를 구조화된 객체로 만들며, 생성된 객체들은 main에서 g_swifood의 데이터 매니저들에 의해 관리된다.


```

class SwifoodLibrary {

public:
    void load(const std::string& fileName);

    IngredientManager ingredientManager;
    ActionManager actionManager;
    RecipeManager recipeManager;

private:

};

```

그림 20. SwifoodLibrary 클래스

3.2.2.4 의미 분석 및 객체 생성(Semantic Analysis & Object Creation)

```

class Recipe {

public:
    Recipe(const RecipeDefinition& definition);

    void print();

private:
    std::string _name;
    std::vector<Ingredient> _ingredients;
    std::vector<Step> _steps;

    std::unordered_set<std::string> _equipments;
    std::unordered_set<std::string> _ingredientInIngredients;
    std::unordered_set<std::string> _ingredientInSteps;
    std::unordered_set<std::string> _relatedRecipes;
    std::vector<Action> _method;

    int _check_ingredients();
    int _check_steps();
    int _check_recipe_error();
    void _analyze_equipments();
    void _steps_to_method();

    void _print(int level);
    void _print_name(int level);
    void _print_equipments(int level);
    void _print_ingredients(int level);
    void _print_method(int level);
    void _print_related_recipes(int level);
    void _print_space(int level);

};

```

그림 21. Recipe 클래스

모든 .swifd 파일의 파싱이 완료되면 각 매니저들에 모든 정의 정보가 저장된 상태가 된다. 그 후 main 함수에서 사용자가 입력한 레시피 이름(argv[1] 값)을 이용해 레시피 정의 정보를 찾은 후 실제 Recipe 객체를 생성한다.

3.2.2.5 오류 처리(Error Handling)

```
class ErrorHandler {
public:
    static ErrorHandler& get_instance();

    void report(const std::string& type, const std::string& target, const std::string& msg);
    bool is_error_exist() const;
    void print_errors() const;

private:
    std::vector<Error> _errorList;
};
```

그림 22. ErrorHandler 클래스

프로그램의 수행 과정 중 발생한 에러들은 ErrorHandler에 기록된다. ErrorHandler는 싱글톤 객체로 생성되어 프로그램 전체의 에러들을 관리하며, 결과 출력 과정 이전에 에러 존재 여부를 파악하고, 에러가 존재하는 경우 에러 메시지를 출력하는 역할을 한다.

3.2.2.6 결과 출력(Output Generation)

```

void Recipe::print()
{
    _print(level: 0);
}

void Recipe::_print(int level)
{
    int checkSum = _check_ingredients() + _check_steps() + _check_recipe_error();

    if (checkSum == 0) {
        _print_name(level);
        std::cout << std::endl;

        _print equipments(level);
        std::cout << std::endl;

        _print ingredients(level);
        std::cout << std::endl;

        _print method(level);
        std::cout << std::endl;

        _print_related_recipes(level);
        std::cout << std::endl;
    }
}

```

그림 23. print 함수 구현

ErrorHandler에 기록된 에러가 존재하지 않다면 레시피의 출력이 수행된다. 출력은 일정한 구조를 가진 텍스트 형태로 stdout을 통해 수행되며, 들여쓰기 정도의 처리를 위해 level을 이용한다.

3.3. 핵심 모듈 구현

3.3.1. 데이터 관리자 구현

```
class ActionManager {
public:
    bool is_defined(const std::string& name) const;
    std::string get_target_type(const std::string& name) const;

    void add_action(const ActionDefinition& action);

    bool check_target(const Action& action) const;
    bool check_required_parameters(const Action& action) const;
    bool check_xor_parameters(const Action& action) const;

    std::vector<std::string> get_required_parameters(const Action& action) const;
    void fill_default_parameters(Action& action) const;

private:
    std::map<std::string, ActionDefinition> _actions;
};
```

그림 24. ActionManager 클래스

```
class IngredientManager {
public:
    void add_ingredients(const std::string& category, const std::vector<std::string>& ingredients);
    bool is_defined(const std::string& name) const;
    std::string get_category(const std::string& name) const;

private:
    std::unordered_set<std::string> _ingredients;
    std::map<std::string, std::string> _ingredientToCategory;
    std::map<std::string, std::vector<std::string>> _categoryToIngredients;
};
```

그림 25. IngredientManager 클래스

```
class RecipeManager {
public:
    bool is_defined(const std::string& name) const;
    void add_recipe(const RecipeDefinition& recipe);
    RecipeDefinition get_definition(const std::string& name) const;

private:
    std::map<std::string, RecipeDefinition> _recipes;
};
```

그림 26. RecipeManager 클래스

파서에서 생성된 Swifood의 정의 정보 객체들은 데이터 매니저들에 의해 관리된다. 데이터 매니저는 각각 액션, 재료, 레시피의 정의 정보를 담당하며, 중복 정의를 비롯하여 정의 정보 객체 생성 과정에서 발생하는 문제들을 담당한다.

3.3.2. 레시피 오류 분석 구현

```
int Recipe::_check_recipe_error()
{
    int checkSum = 0;

    if (_ingredientInIngredients != _ingredientInSteps) {
        std::string undeclaredIngredients = "[ ";
        std::string unusedIngredients = "[ ";
        for (const auto& ingredient: const string &: _ingredientInSteps) {
            if (_ingredientInIngredients.find(k: ingredient) == _ingredientInIngredients.end()) {
                undeclaredIngredients += ingredient + " ";
            }
        }
        undeclaredIngredients += "]";
        for (const auto& ingredient: const string &: _ingredientInIngredients) {
            if (_ingredientInSteps.find(k: ingredient) == _ingredientInSteps.end()) {
                unusedIngredients += ingredient + " ";
            }
        }
        unusedIngredients += "]";

        ErrorHandler::get_instance().report(type: "Recipe", target: _name, msg: "is not complete. Maybe some ingredients are missing or not used.");
        checkSum++;
    }

    return checkSum;
}
```

그림 27. 레시피 오류 검출 기능

레시피 오류 분석 기능은 정의되지 않은 재료의 사용이나 사용되지 않은 재료 검출 등 레시피에 존재하는 문제점들을 찾아내는 역할을 수행한다.

3.3.3. 필요 도구 자동 분석 구현

```
void Recipe::_analyze equipments()
{
    for (const auto& step: Step const &: _steps) {
        if (step.parameters.find(k: "in") != step.parameters.end()) {
            for (const auto& equipment: const string &: step.parameters.at(k: "in").value) {
                _equipments.insert(x: equipment);
            }
        }
        if (step.parameters.find(k: "on") != step.parameters.end()) {
            for (const auto& equipment: const string &: step.parameters.at(k: "on").value) {
                _equipments.insert(x: equipment);
            }
        }
        if (step.parameters.find(k: "with") != step.parameters.end()) {
            for (const auto& equipment: const string &: step.parameters.at(k: "with").value) {
                _equipments.insert(x: equipment);
            }
        }
    }
}
```

그림 28. 필요 도구 자동 분석

필요 도구 자동 분석 기능은 레시피에 정의된 스텝들을 분석하여 레시피에서 사용된 도구들을 찾아내는 역할을 담당한다.

4. 연구 결과 분석 및 평가

4.1. 기능별 결과

4.1.1. 언어 파싱 및 해석

현재 작성된 Swifood 인터프리터는 일부 기능들이 구현되지 않았다. 각 부분별 언어 설계를 만족하지 못하는 기능들은 다음과 같다.

4.1.1.1 Ingredient

재료 카테고리 기능은 이를 사용하여 레시피의 카테고리 분류 등의 기능을 추가하기 위해 준비됐다. 하지만 이를 실제로 처리하는 부분이 기획상으로만 존재하고 언어 설계에 반영되지 못하였다. 또한 인터프리터의 코드에서도 이를 다루기 위한 구조는 어느 정도 고려됐으나 실제로 활용은 되지 않고 있다.

4.1.1.2 Action

Action의 파라미터는 기본 파라미터를 제외한 나머지 부분들이 실제 설계에 부합하지 못하게 구현됐다. 요구 파라미터의 경우 Step에서 호출시에 필수적으로 전달되어야 하며, 이를 만족하지 못한 경우 에러 처리를 해야 하지만 실제 구현은 단순히 요구 파라미터 부분을 삭제하는 방식으로 구현됐다. XOR 파라미터의 경우 이를 처리하는 부분의 구현을 실패하여 언어 명세상으로는 존재하나 실제로는 사용이 불가능하다.

4.1.1.3 Recipe

Recipe의 재료 선언 부분에서 용량은 Amount type 값을 지정하도록 되어 있으며, 이는 SI 단위계의 각 단위와 수치 값을 처리하기 위해 설계됐다. 하지만 실제 구현에서 이는 단순한 문자열 값으로 구현됐다. 이러한 구조는 허용되지 않은 단위 사용 등의 문제를 검출해낼 수 없어 설계에서 의도했던 기능이 정상적으로 동작하지 못할 가능성을 내포하고 있다.

4.1.1.4 Equipment

Equipment는 언어 명세상 별도의 타입으로 존재하며 기획 단계에서 레시피 조리 동작 관련 부분 중 병렬 처리 과정에서 활용될 예정이었다. 하지만 이것이 설계 단계까지 이르지 못하였고, 코드 부분 또한 이와 관련된 준비가 되어있지 않다. 이러한 문제로 인해 현재 구현 단계에서 Equipment는 단순한 문자열로 처리되고 있다.

4.1.2. 오류 검출 및 처리

Swifood의 오류 검출 기능은 작성한 코드의 문제점을 쉽게 파악하고 이를 해결하기 위한 목적으로 설계됐다. Swifood의 ErrorHandler에 의해 관리되며, 코드에 문제가 존재할 경우 레시피 출력의 수행 대신 에러 메세지들이 출력된다. 하지만 현재의 구현은 파싱 에러 발생시에 에러가 발생한 위치를 정확하게 알기 어렵고, 일부 에러 메세지들이 가독성이 좋지 못한 구조로 출력된다.

```
47 Recipe Test_Parse_Error {  
48     Ingredients:  
49     |   Egg 4ea.  
50  
51     Steps:  
52     |   put Egg (in: dish).  
53
```

그림 29. 파싱 에러 유발 코드

```
sunjae@Sunjae-MacBook-Air Swifood % ./swifood Test_Parse_Error  
Error: File 'recipes.swifd' has error in line 53, near '' (syntax error).  
Error: Recipe 'Test_Parse_Error' is not defined.  
2 errors are found.
```

그림 30. 파싱 에러 검출 결과

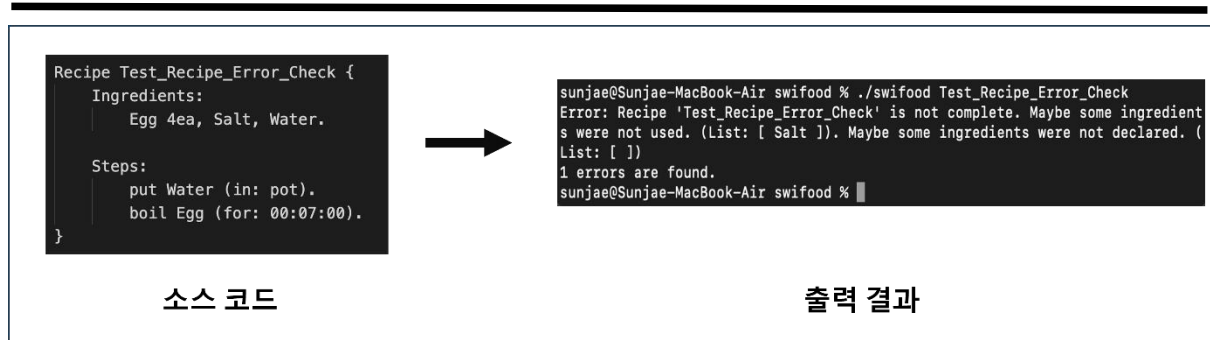


그림 31. 가독성이 좋지 않은 에러 메시지

Recipe에서 자기 자신을 재료로 사용하는 코드는 허용되지 않는다. 이러한 구조는 레시피가 재료로 완성된 자기 자신이 필요하다는 뜻이 되므로 논리적으로 모순이 발생한다. Swifood에서는 이러한 에러를 처리하여 출력 과정에서 무한 재귀 현상 발생을 차단한다.

```
Recipe Test_Using_Self {
  Ingredients:
    Test_Using_Self.

  Steps:
    put Test_Using_Self (in: pot).
}
```

그림 32. 스스로를 재료로 사용하는 레시피 코드

```
sunjae@Sunjae-MacBook-Air Swifood % ./swifood Test_Using_Self
Error: Recipe 'Test_Using_Self' is using itself as ingredient.
Error: Recipe 'Test_Using_Self' has 1 problems in <Ingredients>.
2 errors are found.
```

그림 33. 레시피가 스스로를 재료로 사용하고 있음을 알려주는 에러 메시지

Swifood의 필요 도구 자동 분석 기능은 데이터의 파싱 및 처리 과정에서 자동으로 실행되며, 소스 코드에 오류가 없는 경우 수행되는 레시피 출력의 Equipments 부분에서 결과를 확인할 수 있다.

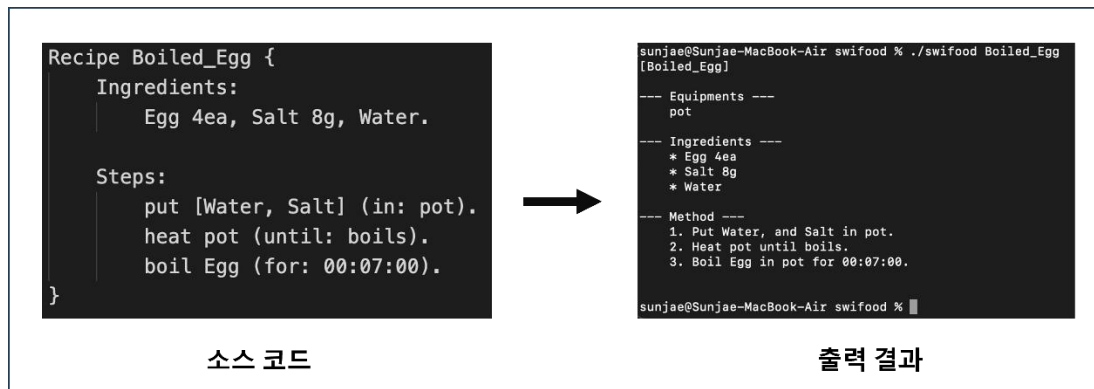


그림 34. 필요 도구 자동 분석 기능 결과

필요 도구 자동 분석 기능은 언어 명세와 비교했을 때 큰 이슈는 없지만 각 도구들이 오직 하나만 존재할 수 있고, 이로 인해 단순히 특정 도구를 사용한다는 수준을 벗어나지 못한다는 문제점이 있다.

5. 결론 및 향후 연구 방향

이번 연구의 과제는 요리 레시피 기술을 위한 도메인 특화 언어 Swifood를 설계하고, 이를 실행시킬 수 있는 인터프리터를 구현하는 것이다. 이를 통해 기존 레시피 기술에 있었던 기술 문법 차이로 인한 파편화 등의 문제들을 개선하였고, 향후 자동화 조리 등에 접목할 수 있도록 하였다. 또한 레시피 오류 검사나 필요 도구 자동 분석과 같이 편의 기능들을 제공하여 기존에 레시피 작성에서 놓치기 쉬운 부분들을 바로 잡기 좋게 하였다.

하지만 구현된 부분들이 현재 내포하고 있는 문제점들도 존재한다. XOR 파라미터를 비롯한 일부 기능들이 구현되지 않았으며, 에러 처리와 레시피 오류 출력 기능 등의 편의 기능들 역시 여러 문제점들을 가지고 있다. 인터프리터 동작에 있어서 최종적으로 출력되는 레시피의 출력 결과가 단순한 구조의 텍스트 포맷 출력 수준에 머물러 있는 점 또한 Swifood가 나아가야 할 길이 아직 많이 남아있음을 보여준다.

Swifood의 향후 연구 방향은 병렬 처리 기능을 비롯한 최초 기획에서 누락된 부분들의 설계와 기존의 언어 명세 중 제대로 구현되지 못한 부분들의 보완, GUI 출력을 비롯한 레시피 출력 포맷의 다양화, 그리고 현재 구현된 편의 기능들이 가지고 있는 문제점들의 개선이라고 생각한다. 또한 Swifood 코드를 자동화 조리에 접목시키기 위한 새로운 인터프리터의 개발도 Swifood가 나아갈 길이라고 생각한다.

자동화 조리의 구현은 쉽지 않은 일이다. Swifood는 여러 조리 동작들이 가지고 있는 표현들의 공통점을 취합하고 단순화함으로써 조리 동작을 추상적으로 표현하고 있다. 이러한 까닭에 실제 자동화 조리에 접목시키기 위해선 자동화 조리 로봇의 동작을 추상화하는 작업이 필요하며, Swifood 코드가 실제로 동작되도록 하기 위해선 그 사이를 이어줄 중간다리가 필요하다. 이러한 작업은 기존의 레시피 출력과 다르게 단순히 언어 설계 뿐만 아니라 실제 자동화 조리 로봇의 동작을 제어하는 지식을 필요로 하기에 성취하기 쉽지 않은 일임을 알 수 있다.

6. 참고 문헌

- [1] FORMATDATA. RecipeML. Available at: <http://www.formatdata.com/recipe/ml/>.
- [2] ROORDA, Auke. Corel: A DSL for Cooking Recipes. 2021. PhD Thesis.
- [3] COOKLANG. Cooklang Specification. Available at: <https://cooklang.org/docs/spec/>.
- [4] Bollini, M., Tellex, S., Thompson, T., Roy, N., & Rus, D. (2013). Interpreting and executing recipes with a cooking robot. In J. Desai, G. Dudek, O. Khatib, & V. Kumar (Eds.), *Experimental Robotics* (pp. 481-495). Springer International Publishing.
- [5] DESIGNBOOM. Moley robotic kitchen: world's first automated kitchen prepares gourmet meals. Available at: <https://www.designboom.com/technology/moley-robotic-kitchen-chef-12-07-2015/>.
- [6] SAMSUNG NEWSROOM. Get a glimpse of the next-generation innovations on display at Samsung's technology showcase. Available at: <https://news.samsung.com/global/get-a-glimpse-of-the-next-generation-innovations-on-display-at-samsungs-technology-showcase>.
- [7] Noh, D., Nam, H., Gillespie, K., Liu, Y., & Hong, D. (2024). YORI: Autonomous Cooking System Utilizing a Modular Robotic Kitchen and a Dual-Arm Proprioceptive Manipulator.
- [8] <https://danielarus.csail.mit.edu/index.php/2015/10/robots-and-cooking-bakebot/>
- [9] <https://robbreport.com/gear/electronics/moley-robotics-robot-kitchen-uk-for-sale1234590791/>
- [10] <https://news.samsung.com/global/get-a-glimpse-of-the-next-generation-innovations-on-display-at-samsungs-technology-showcase>
- [11] <https://spectrum.ieee.org/romela-cooking-robot>

-
- [12] Fowler, Martin; Parsons, Rebecca. "Domain Specific Languages"
 - [13] <https://gcc.gnu.org/>
 - [14] Stroustrup, Bjarne (2010, March 7) https://www.stroustrup.com/bs_faq.html#invention
 - [15] Stroustrup, Bjarne (1997). "1". The C++ Programming Language (Third ed.). Addison-Wesley. ISBN 0-201-88954-4
 - [16] Levine, John (August 2009). flex & bison. O'Reilly Media. p. 9. ISBN 978-0-596-15597-1.
 - [17] <https://www.gnu.org/software/bison/>
 - [18] Feldman, S. I. (April 1979). "Make --- A Program for Maintaining Computer Programs". Software: Practice and Experience. 9 (4): 255-265.