9. Executing Rewriting Logic Specifications in Maude

이 장은 **Maude에서 재작성 논리(rewriting logic) 명세를 실행하는 방법**을 소개한다.

- **등식 명세(equational specification)**는 종료성과 합류성이 보장된다고 가정하며, 목표는 식을 단순화해 **정규형(normal form)**을 계산하는 것이다. 어떤 식을 적용할지 고민하지 않고 가능한 모든 식을 적용한다.
- 반면 **재작성 명세(rewrite specification)**는 시스템의 **모든 가능한 동작(behavior)**을 모델링하며, 종료되거나 합류적이지 않을 수도 있다.

Maude에서는 이러한 재작성 명세를 다음과 같이 실행할 수 있다:

1. `rew` / `frewrite` **명령어**: 초기 상태에서 시작해 재작성 규칙을 적용하여 시스템의 한 가지 가능한 동작을 시뮬레이션한다. 종료되지 않을 수도 있으므로 사용자가 단계 수의 상한을 설정할 수 있다.
2. `search` **명령어**: 너비 우선 탐색(BFS)을 이용해 초기 상태로부터 특정 상태 패턴이 **도달 가능한지** 확인한다.

즉, 이 장은 **Maude가 동적 시스템의 동작을 실행/탐색하여 분석하는 방법**을 다루며, 16장에서 다룰 **시간 논리(temporal logic)** 분석의 기초가 된다.

9.1 Executing One Sequential Rewrite Step

Maude의 재작성 실행 방식

- 재작성 논리(rewriting logic)는 이론적으로 여러 위치를 **동시에(concurrently)** 바꿀 수 있지만, Maude는 실제로 **하나씩 순차적으로(one-step sequential)** 바꾼다.
- 하지만 손실은 없다. 왜냐하면 모든 병렬 재작성은 여러 개의 순차 재작성으로 **분해 가능하기 때문**이다. (Proposition 8.2)

재작성 규칙 실행의 본질

- 등식이 없는 경우, 재작성 규칙을 적용하는 것은 등식을 적용하는 것과 같다.
- 하지만 등식이 있는 경우, 어떤 항 t에 규칙 l → r이 적용 가능한지를 판단하는 것은 "(E ⊢ lσ = t)"를 확인하는 문제로 귀결된다. (즉, t가 l과 **E-동등한지**를 확인해야 함)
- 이 문제는 일반적으로 **결정 불가능(undecidable)** 하다.

Maude의 해결 방법

Maude는 다음의 절차를 따른다:

1. **E-정규형(normal form)**으로 변환

   → 등식들을 모두 적용해 항을 단순화시킨다.

2. **규칙 적용 여부 검사**

   → 단순화된 항 t!에 대해 규칙의 왼쪽 항이 일치하는지 확인한다.

3. **규칙 적용 및 결과 정규화**

   → 결과 t'를 다시 등식으로 정규형 t'!으로 만든다.

즉, Maude는 "항을 정규형으로 만든 뒤" → "재작성 규칙을 적용" → "다시 정규형으로 변환" 과정을 거친다.

생성자 항(constructors) 규칙의 필요성

- 이렇게 정규형을 먼저 만드는 방식에서는 규칙이 **사라질 위험**이 있다.

  → 즉, Maude가 규칙을 "놓칠(miss)" 수 있음.

- 따라서 **규칙의 왼쪽 항은 항상 생성자 항이어야 한다.** (정의된 기호(+, -, f 등)를 포함하지 않아야 함)

예시 요약

- Example 9.1

```
eq a = b .
eq b = c .
rl [1] : b ⇒ d .
```

  - `a`, `b`, `c`는 모두 같지만, Maude는 `a ⇒ d` 또는 `c ⇒ d`를 자동으로 보지 못함.
  - 왜냐하면 `a`를 `b`로 정규화하고 나서야 규칙을 적용하기 때문임.

- Example 9.2
  - `rl [1] : b ⇒ d` 대신 `rl [1] : c ⇒ d` 로 바꿔야 함.
  - 그래야 Maude가 정규화 후에도 규칙을 "놓치지 않음."

- Example 9.3

```
person("Edward", 21+11, single)
```

  → Maude는 먼저 `21+11`을 계산해서 `32`로 만든 다음, `person("Edward", 32, single)`에 대해 규칙을 적용한다.

  → 결과적으로 `person("Edward", 33, single)`이 됨.

- Example 9.4

```
rl : person(X, N + 1, S) ⇒ person(X, N, S) .
```

  - 이 규칙은 작동하지 않음. 왜냐하면 Maude가 먼저 `N + 1`을 계산해서 정규형으로 만들기 때문에 규칙의 왼쪽 패턴이 더 이상 일치하지 않음.
  - 해결책:

```
rl : person(X, s N, S) ⇒ person(X, N, S) .
```

    (`s`는 자연수의 생성자, 즉 successor 기호)

조건부 재작성 (Conditional Rewriting)

- 개념 요약

  Maude에서 **조건부 재작성 규칙**은 다음과 같은 형태를 가진다.

```
crl [ruleName] : left ⇒ right if Condition .
```

  - 규칙을 적용하기 전에 `Condition`이 **참일 때만** 재작성 수행.

- 즉, “단순히 바꾸는 것”이 아니라 “조건을 만족해야 바꾸는 것.”

- 조건의 형태

  대입형 조건 `x := t`

  - 의미: 변수 `x`를 식 `t`의 **정규형(normal form)** 결과로 치환(instantiation).
  - 수학적으로는 `x = t`와 같지만, **Maude에서는 실제 계산을 수행하여 x에 값을 대입함.**

    ```
    crl [birthday] : person(X, N) ⇒ person(X, M)
       if M := N + 1 .
    ```

    → `M`은 `N + 1`의 결과로 대체되어 `person("Edward", 33)` 등으로 재작성됨.

- **도달성 조건** `u ⇒ u'`
  - 의미: `u`가 **하나 이상의 재작성 단계를 거쳐** `u'`**로 변환될 수 있는가?**
  - 단순히 `u`를 계산(normalize)하는 게 아니라, Maude는 `u`로부터 가능한 모든 재작성 경로를 **탐색(breadth-first search)** 한다.
  - 만약 `u'`에 도달하지 못하면, Maude는 **무한 탐색**에 빠질 수도 있음.

- 조건의 평가 순서
  - 여러 조건이 있을 경우 **왼쪽에서 오른쪽으로 차례로 평가**.
  - 각 조건에서 사용되는 변수는 반드시:
    1. 규칙의 왼쪽 항에 있거나,
    2. 앞선 조건에서 이미 정의되어 있어야 한다.
  - 예시:

    ```
    crl [ex] : f(X, Y) ⇒ g(X, Y)
       if A := X + 1 /\ B := A + Y /\ C := B * 2 .
    ```

    → `A` → `B` → `C` 순서로 평가해야 함.

- 종합 동작 과정

  Maude가 조건부 규칙을 적용할 때의 절차 :

  1. 왼쪽 항(`left`)과 현재 항(`t`)을 일치시킴 (matching)
  2. 각 조건을 왼쪽에서 오른쪽으로 평가:
     - `x := t` → 변수에 정규형 결과를 대입
     - `u ⇒ u'` → BFS로 도달 가능성 탐색
  3. 모든 조건이 참이면 `right`로 재작성
  4. 결과 항을 다시 **E-정규형**으로 변환 (등식 부분 적용)

- 주의점
  - `u ⇒ u'` 조건은 단순 계산이 아니라 **탐색 기반**으로 판정된다.
  - 따라서 잘못된 규칙 설계는 **무한 탐색 루프**를 초래할 수 있다.
  - 효율적이고 안전한 규칙 설계를 위해서는:

- **조건 수를 최소화**
- `x := t` **중심으로 표현**
- **정의된 기호 대신 생성자(constructor) 중심의 좌변 사용**

한 줄 요약

- Maude는 등식 명세를 통해 항을 정규형으로 단순화한 뒤, 생성자 항을 기준으로 한 단계씩 순차 재작성을 수행한다.
- 규칙의 좌변은 생성자 항이어야 하며, 조건부 재작성은 탐색 기반으로 판정된다.

9.2 Simulating Single Behaviors

`rew` / `frew` **명령어의 역할**

- Maude에서 **시스템의 단일 동작(single behavior)**을 "실행(simulate)"하는 명령.
- `rew`: rewrite rules을 순차적으로 한 단계씩 적용.
- 규칙을 적용하다가
  - 더 이상 적용할 규칙이 없거나,
  - 사용자가 지정한 rewrite 횟수 제한에 도달하면

  실행이 멈춘다.
- 스펙이 비종결(nonterminating)이라면, 제한이 없을 경우 **무한히 실행**될 수 있다.

실행 과정의 일반적인 형태

- 모든 항은 먼저 식(E) 정규형(E-normal form)으로 줄여진 뒤 규칙이 적용됨.
- 실행 순서 표현:

  t ⇝* t! → t1 ⇝* t1! → t2 ⇝* t2! → ... → tn ⇝* tn!

`rew` **명령어의 문법**

```
rew t .
rew [n] t .
```

- `t` : rewrite할 항(term)
- `n` : rewrite를 수행할 최대 단계 수
- `frew` : 동일한 문법이지만 "공정한(fair)" 재작성 방식
- `set trace on` 하면 각 rewrite 단계가 전부 출력됨

예시

```
Maude> rew [100] in ONE-PERSON : person("Peter", 46, married) .
result Person: person("Peter", 146, married)
```

→ `person` 객체의 나이를 100번 증가시키는 예시. (`in` 키워드는 어떤 모듈에서 rewrite할지 지정)

규칙 선택의 차이 (`rew` vs `frew`)

항목	`rew`	`frew`
규칙 선택	“라운드로빈(round-robin)” 방식	동일
적용 위치 우선순위	**항의 위쪽(top)**, **왼쪽(leftmost)** 서브항 우선	모든 서브항에 **공정(fair)** 하게 분배
결과 일관성	비결정적일 수 있음 (규칙 순서 의존)	결정적 (항상 같은 결과)

예제 분석

- TEST-REW1
  - 세 규칙이 **같은 패턴**을 공유:

    ```
    f(rule1(N), rule2(M), rule3(K)) ⇒ ...
    ```

    - 한 항에서 세 규칙 모두 적용 가능 → Maude가 라운드로빈으로 적용
    - 결과: `rule1` 34회, `rule2` 33회, `rule3` 33회 적용
- TEST-REW2
  - 각 규칙이 **서로 다른 서브항**에만 적용 가능:

    ```
    rule1(N) ⇒ rule1(s N)
    rule2(N) ⇒ rule2(s N)
    rule3(N) ⇒ rule3(s N)
    ```

    - `rew`는 항상 **왼쪽부터(top-leftmost)** 탐색하므로 첫 번째 `rule1`만 계속 적용됨.
    - 결과: `rule1` 100회, 나머지 0회
    - 반면 `frew`는 모든 서브항에 대해 공정하게 적용하므로 각 규칙이 번갈아 실행됨.

9.3 Search

목적

- `rew`나 `frew`는 **한 가지 실행 경로**만 보여주기 때문에, 시스템 전체의 모든 가능한 동작을 확인하기엔 부족하다.
- 그래서 Maude는 `search` **명령**을 제공해, **초기 상태에서 도달 가능한 모든 상태**를 **탐색(breadth-first)** 한다.

탐색 방식

- **너비 우선 탐색(BFS)** : 1단계에서 reachable한 상태 → 2단계 → 3단계 ... 순서대로 확장.
- 이미 방문한 상태는 다시 탐색하지 않음.
- 도달 가능한 상태가 **무한하면 종료되지 않을 수 있음.**

기본 명령 형식

```
search t0 ⇒ pattern .
```

```
search t0 ⇒ pattern such that cond .
```

- `t0` : 초기 상태
- `pattern` : 찾고 싶은 형태(변수 포함 가능)
- `cond` : 조건식 (eq의 조건 형태와 동일)

화살표 의미

화살표	의미
`⇒1`	정확히 1단계에서 도달 가능한 상태
`⇒*`	0회 이상 단계에서 도달 가능한 상태
`⇒+`	1회 이상 단계에서 도달 가능한 상태
`⇒!`	더 이상 rewrite 불가능한 **종료 상태**

활용 예시

- 어떤 사람이 한 단계 후 어떤 상태가 되는지 → `=>1`
- 나중에 나이가 몇이 될 수 있는지 → `=>*`
- 조건(`such that`)으로 제한 가능 (`N < 32` 등)
- 최종 상태(더 이상 변화 없음) → `=>!`

결과 해석

- Maude는 각 해(solution)를 "상태 번호"와 함께 출력함.
- `show path n` → 초기 상태에서 `n` 번째 상태까지의 **rewrite 경로** 출력
- `show path labels n` → 적용된 **규칙(label)** 만 출력

주의점

- 상태가 무한히 많으면 탐색이 끝나지 않을 수 있음.
- 그래서 다음처럼 **탐색 개수(n)** 또는 **rewrite 단계 제한(d)** 을 둘 수 있음:

```
search [n] ...     -- n개의 해만 찾음
search [,d] ...    -- 최대 d단계까지만 탐색
search [n,d] ...   -- 둘 다 제한
```