2. Equational Specification in Maude

본 장에서는 Maude에서 데이터 타입을 **방정식적 명세(equational specification)**로 정의하는 방법을 다룬다.

Maude는 선언적 프로그래밍 언어로서 "무엇을 계산할지"를 정의하며, 명령형 언어(Java, C 등)의 "어떻게 계산할지"와 대비된다.

2.1 Hello World : Our First Maude Specifications

- 자연수와 불리언 값을 정의하는 간단한 명세를 통해 Maude 실행 방식을 소개한다.
- 방정식 명세는 **함수형 모듈(functional module)** 구조를 가지며, 정렬(sorts), 함수 기호(operators), 변수, 방정식으로 구성된다.
- 구문 예시:

```
fmod MODULENAME is
  BODY
endfm
```

- 주석은 `***` 또는 `---` 기호로 작성 가능하다.

2.1.1 Natural Numbers with Addition

```
fmod NAT-ADD is
    sort Nat .
    op 0 : -> Nat [ctor] .
    op s : Nat -> Nat [ctor] .
    op _+_ : Nat Nat -> Nat .

    vars M N : Nat .

    *** Define the addition function recursively :
    eq 0 + M = M .
    eq s(M) + N = s(M + N) .
endfm
```

- 정렬(sort) = 자료형(type)
  - 프로그래밍 언어에서 int, bool 같은 자료형이 있듯이, Maude에서는 이를 정렬(sort)이라 부른다.
  - 즉, `sort Nat .`은 Nat이라는 이름의 새로운 데이터 타입(정렬)을 선언한 것이다. (이 타입은 자연수를 나타냄)
- Nat 정렬의 값 (data values)
  - Nat 정렬을 만들면, 그 안에 값(value)들을 정의해야한다.
  - `op 0 : -> Nat [ctor] .` : 0은 자연수 0을 의미하는 상수 (ctor : constructors의 약자로 생성자)
  - `op s : Nat -> Nat [ctor] .` : successor 함수. 즉, s(N)은 **N의 다음수**

- Maude에서 변수는 값 할당이 아니라, 방정식이 왼쪽에서 오른쪽으로 적용되어 식을 단순화하는 과정으로 일반적으로 표현하기 위한 자리 표시자이다.

- nat-add.maude로 저장할 수 있고, in nat-add.maude로 command에서 실행할 수 있다.

```
Maude> in nat-add.maude
```

- 규칙 :
  - Maude는 대소문자를 구분한다. 즉, `Nat`과 `nat`은 서로 다른 종류이다.
  - 각 선언은 반드시 공백 + 마침표(`.`)로 끝나야 한다. 단, `endfm` 뒤에는 마침표가 오면 안 된다.
  - 중위(infix) 연산자(예: `+`)를 사용할 때는 앞뒤에 반드시 공백이 있어야 한다.
    - 올바른 예: `eq 0 + M = M .`
    - 잘못된 예: `eq 0+M = M .`
  - `_`와 `+` 사이에는 공백이 있으면 안 된다. (즉, `op _+_ : Nat Nat -> Nat .` 처럼 써야 한다)
- Maude를 종료하려면 `q` (또는 `quit`) 명령어를 입력한다.

- `red` (혹은 `reduce`) 명령어는 주어진 표현식의 "값"을 계산한다. 예를 들어 `2 + 3`은 방정식을 왼쪽에서 오른쪽으로 적용하며, 더 이상 규칙을 적용할 수 없을 때까지 단순화된다.
- 예시 :

```
Maude> red s(s(0)) + s(s(s(0))) .
```

  - in nat-add.maude (이 모듈에 정의된 규칙들을 불러와서 앞으로 계산할 때 적용)을 해서 계산이 가능한 것
- 결과 :

```
Maude> red s(s(0)) + s(s(s(0))) .
reduce in NAT< : s(s(0)) + s(s(s(0))) .
rewrites: 3 in 0ms cpu (0ms real) (3000000 rewrites/second)
result Nat: s(s(s(s(s(0)))))
```

  - 여기서 3은 결과를 계산하기 위해 적용한 rewrite(방정식 적용) 횟수임.
    - 왜 3인가 ?

      바깥쪽 s(M) + N → 안쪽 s(M) + N → 마지막 0 + N

2.1.2 The Boolean Values and Functions

```
fmod BOOLEAN is
  sort Boolean .
  ops true false : -> Boolean [ctor] .
  op not_ : Boolean -> Boolean [prec 53] .
  op _and_ : Boolean Boolean -> Boolean [prec 55] .
  op _or_ : Boolean Boolean -> Boolean [prec 59] .
```

```
    var B : Boolean .
    eq not false = true .   eq not true = false .
    eq true and B = B .     eq false and B = false .
    eq true or B = true .    eq false or B = B .
endfm
```

- 정렬과 연산자의 실제 이름은 중요하지 않다.
- 1차 논리는 함수 기호 사이의 우선순위(precedence)이다.
  - e.g, 부정이 연결보다 우선순위가 높기에 not x and y는 (not x) and y가 된다.
- 우선순위 `prec n`을 부여할 수 있다. 숫자가 낮을 수록, 우선순위는 높다.

2.1.3 Module Importation

```
fmod NAT< is
    protecting NAT-ADD .
    protecting BOOLEAN .

    op _<_ : Nat Nat -> Boolean .

    vars M N : Nat .

    eq 0 < s(M) = true .
    eq M < 0 = false .
    eq s(M) < s(N) = M < N .
endfm
```

- 모듈은 이미 maude에 들어와있는 다른 모듈을 `protecting` or `including`을 통해 가지고 올 수 있다.
- `eq 0 < s(M) = true .` : 0은 어떤 자연수의 successor(후행, 즉 1이상)보다 항상 작다는 의미
- `eq M < 0 = false .` : 어떤 수 M도 0보다 작을 수 없다는 의미

2.2 Many-Sorted Equational Specifications

Signature (서명)

- **정의**: 어떤 연산 기호들이 있고, 그 연산들이 어떤 **sort(자료형)** 을 입력/출력으로 가지는지 정의한 것.
- **형식**: Σ = { f : w → s }
  - w : 입력 sort들의 시퀀스
  - s : 출력 sort
- **예시**:
  - `0 : -> Nat` (자연수 0)
  - `s : Nat -> Nat` (successor 함수, 즉 +1)
  - `+ : Nat Nat -> Nat` (덧셈)

Variables (변수)

- **정의**: 각 sort에 대해 무한히 많은 변수를 가질 수 있음.
- **예시**:
  - `X : Nat, Y : Nat`

Terms (항)

- **정의**: signature의 함수 기호와 변수들을 이용해 만들어지는 식.
- **조건**: 결과 sort가 명확해야 함.
- **예시**:
  - `0, s(0), s(s(0)), X + 0, s(X) + Y`

Ground Terms (기저 항)

- **정의**: 변수가 **없는** term. (즉, 실제 값만으로 구성된 항)
- **예시**:
  - `0, s(0), s(s(0)), 0 + s(0)`

Equations (방정식)

- **정의**: 두 term이 동일하다고 정의하는 것.
- **형식**:
  - 비조건식: $(\forall X)\ t = t'$
  - 조건식: $(\forall X)\ (u_1 = v_1 \wedge \ldots \wedge u_n = v_n) \Rightarrow t = t'$
- **예시 (Maude 문법)**:
  - `eq X + 0 = X .`
  - `ceq max(M, N) = N if M < N = true .`

Many-Sorted Equational Specification

- **정의**: (S, Σ, E) 튜플
  - S : sort들의 집합
  - Σ : signature
  - E : 방정식들의 집합 (eq + ceq)
- **예시**:

```
sort Nat .
op 0 : → Nat .
op s : Nat → Nat .
op _+_ : Nat Nat → Nat .
eq 0 + M = M .
eq s(N) + M = s(N + M) .
```

2.3 Requirements of Equational Specifications

2.3.1 One-to-one Constructor Basis (1대1 생성자 기반)

- **개념**: 데이터 타입의 각 요소는 반드시 **하나의 생성자 ground term**으로 표현되어야 한다.
- 중복 표현(여러 방식으로 같은 원소 표현)이나 쓸모없는 생성자 항("junk")이 있으면 안 됨.
- **예시**: 자연수
  - 0, s(0), s(s(0)), ... 와 같이 정확히 한 가지 방식으로만 표현.
- **한 줄 요약**: **도메인의 모든 값은 생성자 항으로 1:1 대응해야 한다.**

2.3.2 Termination (종결성: 무한 계산 금지)

- **개념**: 어떤 ground term도 무한히 계산되지 않고 반드시 끝나야 한다.
- 잘못된 예시:

```
eq a = b .
eq b = a .
```

  → a ↔ b ↔ a ... 무한 반복.
- 체크 포인트: 재귀 정의는 반드시 어떤 인자가 줄어들어야 함.
- **한 줄 요약**: **항상 계산이 끝나서 멈춰야 한다.**

2.3.3 Uniqueness of the Result (결과의 유일성)

- **개념**: 함수는 입력마다 하나의 값만 가져야 함.
- 방정식 적용 순서에 따라 결과가 달라지면 안 됨.
- **예시**:
  - 올바른 경우: s(s(0 + s(0))) + 0 → 항상 s(s(s(0))).
  - 잘못된 경우: eq a = b, eq a = c → 결과가 b일 수도, c일 수도 있음.
- **관련 개념**: 이를 **합류성(confluence)** 이라고 부름.
- **한 줄 요약**: **계산 경로와 상관없이 결과는 항상 같다.**

2.3.4 Definedness (정의성: 결과는 생성자 항이어야 함)

- **개념**: 모든 계산 결과는 반드시 **생성자 ground term**이 되어야 한다.
- 잘못된 경우: NAT-ADD에서 `0 + M = M` 방정식을 빼먹으면, 더 이상 줄일 수 없는 비-생성자 항이 남는다.
- **요구**: 모든 비-생성자 함수는 모든 생성자 입력에 대해 정의되어야 한다.
- **예시**:
  - 0 + M → M
  - s(M) + N → s(M + N)
- **한 줄 요약**: **최종 계산 결과는 반드시 생성자 항이어야 한다.**

2.3.5 Maude and the Requirements

- **개념**: Maude는 위 요구사항들을 자동으로 보장하지 않는다.

- 첫 번째(1대1 생성자) → 아예 확인 불가.
- 나머지 세 가지(종결성, 합류성, 정의성) → **일반적으로 결정 불가능**.
- 따라서 외부 도구(termination checker, confluence checker 등)를 활용해야 함.
- **한 줄 요약**: **Maude가 대신 체크해주지 않으니, 사용자가 보장해야 한다.**

2.4 Many-Sorted Specification of Data Types

2.4.1 Defining Functions: Getting Started

- 자동으로 함수를 정의하는 방법은 없지만, 빠르게 시작할 수 있는 하나의 힌트는 함수 op f : s → s′를 각 생성자마다 하나 이상의 방정식으로 정의하는 것임.
- 예: 자연수에 대한 `double` 함수 (두 배 함수)를 `0`과 `s(N)`에 대해 정의.
- 두 인자를 받는 함수는 case-by-case 정의 가능.
  - 예: 곱셈 은 첫 번째 인자에 대해 정의:
    - `0 * N = 0`
    - `s(M) * N = N + (M * N)`

2.4.2 Expressiveness of Many-Sorted Equational Specifications

- Bergstra와 Tucker는 `0`과 `s`만으로 제곱(square) 함수를 정의하는 것은 불가능함을 증명.
- 보조 함수(덧셈, 곱셈 등)를 도입하면 제곱과 거듭제곱 정의 가능.
- **표현력**: 보조 함수들을 통해 원하는 모든 계산을 정의할 수 있음.
- **Theorem 2.1**: 모든 계산 가능한 대수는 종료하고(confluent) 유한(finitary)한 다정(sorted) 등식 사양으로 정의 가능.
- → 즉, 어떤 프로그래밍 언어로 할 수 있는 것은 Maude로도 가능.

2.4.3 Maude Specifications of Some Data Types

(1) Lists

- 리스트 정의: `nil`, `__` (리스트 뒤에 원소를 추가하는 생성자).
- 예: `1 2 3` 리스트는 `nil s(0) s(s(0)) s(s(s(0)))`.
- 주요 함수들: `length`, `concat`, `insertFront`, `first`, `last`, `rest`, `reverse`, `remove`, `max`, `isSorted`, 등.
- `first(nil) = 0`처럼 정의되지 않는 경우에는 기본값을 넣어 partial function 처리.

(2) Binary Trees

- 이진트리 정의:
  - `empty : → BinTree`
  - `bintree : BinTree Nat BinTree → BinTree`
- 의미: `bintree(t, n, t')`는 루트가 `n`이고 왼쪽 서브트리가 `t`, 오른쪽 서브트리가 `t'`.
- 예: 그림의 트리(4 루트, 오른쪽 자식 7)는 Maude term으로 표현.
- 지원 함수: `preorder`, `inorder`, `postorder`, `weight`, `size`, `isSearchTree`, `reverse`.

(3) Sets

- 집합(Set)과 다중집합(Multiset)은 중요한 자료형.
- 하지만 `{a, b}`와 `{b, a}`는 동일 집합이라 일대일 생성자 표현이 어려움.
- 예: `{0,1}`은 `empty ; 0 ; s(0)` 또는 `empty ; s(0) ; 0` 두 가지 표현 가능.
- Maude에서는 동치류(equivalence class)를 사용하여 동일 집합을 표현.

2.5 Order-Sorted Equational Specifications

- **Order-Sorted Specification**: 여러 정렬(sort) 사이에 **부분순서(≤)** 를 두어, 한 정렬이 다른 정렬의 부분집합(=subsort)임을 표현.
- 목적:
  - 함수 정의 시 **부분적으로 정의되는 함수(partial function)** 를 잘 다루기 위함.
  - `Nat` (자연수)와 `Int` (정수)처럼 관련 있는 정렬들을 유연하게 연결.
- **Subsort 선언**:

  ```
  subsort Nat < Int .
  ```

  → `Nat`은 `Int`의 부분정렬임을 의미 (모든 자연수는 정수).
- **Order-Sorted Signature (Def. 2.7)**:

  (S, ≤, Σ) = 정렬 집합 S + 부분순서 ≤ + 함수 기호 선언 Σ.
- **Terms in Order-Sorted Signatures (Def. 2.8)**:

  부분정렬 포함 관계에 따라, 더 작은 sort의 term은 더 큰 sort의 term으로 포함된다.

2.5.1 Examples of Order-Sorted Equational Specifications

2.5.1.1 Partiality

- **문제**: 자연수에서 나눗셈은 부분함수(`n / 0`은 정의 안됨).
- **해결**: `NzNat` (non-zero natural)이라는 subsort 정의 → 분모가 항상 0이 아닌 경우만 허용.
- 리스트도 같은 방식으로 `NeList` (non-empty list)를 정의 → `first`, `last`, `rest`, `max` 함수가 항상 total function이 되도록 보장.

2.5.1.2 Constructors for the Integers

- **목표**: 모든 정수가 정확히 하나의 생성자(constructor) ground term으로 표현되도록.
- **구조**:
  - `Zero` = 0
  - `NzNat` = 양의 자연수
  - `NzNeg` = 음의 정수
  - `Nat`, `Neg`, `Int`는 각각 자연수, 음수, 정수 전체를 의미
- **예시**:

- `s(0)` = 1
- `-s(s(0))` = -2
- `p(p(0))` = -2 (전임자 함수 p 이용)

- **연산 정의**: 정수 덧셈/뺄셈을 단계적으로 정의 (Nat → Int 확장).

2.5.1.3 Elements in a List

- **문제**: 리스트를 항상 `nil` 로 시작하면 불편.
- **해결**: 자연수를 (non-empty) 리스트로도 간주.

```
subsort Nat < NeList < List .
```

- 결과: `nil n1 n2 ... nk` 대신 `n1 n2 ... nk` 형태로 깔끔하게 리스트 표현 가능.

2.5.1.4 "Undefined" Values

- **상황**: 때때로 **에러 값**이나 **초기화되지 않은 값**이 필요.
- **해결**: `DefNat` supersort를 정의하고, 그 안에 `noNat` 같은 상수를 추가.
  - 즉, `Nat` ⊂ `DefNat`, 그리고 `DefNat`에 특별 값 `noNat` 존재.
- 이렇게 하면 프로그램 내에서 "정의되지 않은 값"을 안전하게 다룰 수 있음.

2.6 Order-Sorted Equational Specifications

Order-Sorted의 한계

- Order-sorted 사양은 **일부 의미적(subsemantic) 제약**을 표현하기 어려움.
- 예시:
  - **이진 탐색 트리(binary search tree)**: `insertSorted` 같은 연산은 탐색 트리에만 의미 있음.
  - **SortedList**: `insertSorted`, `merge` 같은 연산은 정렬된 리스트에만 정의되어야 함.
- 하지만 이런 "semantic" 제약은 order-sorted만으로는 정의 불가.

Membership Equational Logic의 도입

- 해결책: **membership equational logic**

  → 특정 조건을 만족해야만 어떤 항(term)이 특정 sort에 속한다고 선언 가능.

Membership 공리 형식

```
mb t : s .
cmb t : s if cond .
```

- `t`가 sort `s`에 속함을 선언.
- `cmb`는 조건(`cond`)을 만족할 때만 속함을 허용(conditional membership).

예시: SortedList 정의

```
fmod SORTED-LIST-NAT1 is protecting LIST-NAT1 .
  sort SortedList .
  subsort SortedList < List .
  var L : List .
  cmb L : SortedList if isSorted(L) = true .
endfm
```

- `isSorted(L) = true`일 때만 `L`을 `SortedList`로 인정.

오류와 "Benefit of Doubt"

- `s(s(0)) / (s(0) - 0)` 같은 식은 원래 sort를 가질 수 없음.
- 하지만 Membership equational logic은 이를 **일단 허용하고(error sort)** 가능한 계산을 수행.
- 결과가 정상적인 sort에 도달하면 그때는 유효(term 인정).
- 결과가 여전히 잘못된 경우(`s(0)/0`)는 error sort [Int]에 속함.

Kind 개념

- 각 연결 성분(component)마다 **kind** 존재.
- 예: `[Int]` → Int와 관련된 sort들의 kind.
- sort가 없는 항도 kind에는 속할 수 있으며, 이를 **error term**이라 부름.
- Maude는 자동으로 **kind 기반 선언** 추가:

```
op f : [s1] ... [sn] -> [s] .
```

2.7 Built-in Data Types

- Maude는 자연수, 정수, 유리수, 부동소수점, 문자열, 불리언 값 등 **기본 자료형**을 제공함.
- C++로 효율적으로 구현되어 큰 수 연산, 문자열 처리 등이 가능.
- 일반적인 언어와 달리 **unbounded**(무한정 크기) 자연수, 정수, 유리수를 지원.
- 이들은 `prelude.maude` 파일에 정의되어 있으며, 필요시 수정 가능.
- Boolean은 기본으로 포함되지만, 다른 자료형은 `set include NAT on` 등을 통해 명시적으로 import해야 함.

2.7.1 Booleans

- 불리언 값: `true`, `false`
- 주요 연산:
  - 논리연산: `_and_`, `_or_`, `_xor_`, `not_`, `_implies_`
  - 조건문: `if_then_else_fi`
  - 비교연산: `_==_`, `_=/=_`
- `assoc`, `comm` 속성: 결합법칙, 교환법칙 지원.
- 예시: `ceq M monus N = 0 if M <= N .`

2.7.2 Natural Numbers

- 자료형: `Zero`, `NzNat`, `Nat`
- 생성자: `0`, `s_` (successor)
- 주요 연산:
  - 덧셈: `_+_`
  - 곱셈: `_*_`
  - 나눗셈: `_quo_`, 나머지: `_rem_`
  - 최대공약수: `gcd`, 최소공배수: `lcm`
  - 비교연산: `_<=_`, `_>_`, `_divides_`
- 뺄셈 대신 대칭 차(`sd`) 제공.
- 팩토리얼 예시:

```
eq 0 ! = 1 .
eq (s N) ! = s N * (N !) .
```

2.7.3 Integers

- 정수는 자연수에서 확장.
- 음수 표현: `s 0`, `2009`, `1`, ...
- 주요 연산:
  - 음수 생성자: `-_`
  - 덧셈: `_+_`
  - 뺄셈: `_-_`
  - 절댓값: `abs`

2.7.4 Rational Numbers

- 유리수: `NzRat`, `PosRat`, `Rat`
- 주요 연산:
  - 나눗셈 `/`
  - 소수점 관련 함수: `trunc`, `floor`, `ceiling`, `frac`
- 예시:

```
eq floor(N / M) = N quo M .
eq ceiling(N / M) = ((N + M) - 1) quo M .
```

2.7.5 Floating-Point Numbers

- IEEE-754 **64-bit double precision** 지원.
- 주요 연산: `sqrt`, `log`, `sin`, `cos`, `asin`, `acos`, ...

- 상수 표현: `1.0`, `9.87654321`, `1.23e+14`
- 특별 상수: `Infinity`, `Infinity`
- 예시:

```
red 3.45e+223 * 2.99e+210 .
result Float: Infinity
```

2.7.6 Strings

- 문자열 자료형: `String`, `Char`, `FindResult`
- 주요 연산:
  - `ascii`, `char`
  - 문자열 합치기: `_+_`
  - 길이: `length`
  - 부분 문자열: `substr`
  - 찾기: `find`, `rfind`
  - 비교: `<`, `<=`, `>`, `>=`
- 변환 모듈 `CONVERSION`: 숫자 ↔ 문자열 변환
  - 예시: `string(123, 10) = "123"`, `string(5, 2) = "101"`

2.7.7 Random Numbers

- 의사 난수 생성: `random(k)` (0 이상 2^32 - 1 이하)
- 같은 `k`에 대해 항상 같은 값 반환.
- 범위 제한 예시 (1~100): `(random(k) rem 100) + 1`
- 예시 실행:

```
red random(1) .
result NzNat: 2546248239

red (random(2) rem 100) + 1 .
result NzNat: 34
```