Constructing Canonical Term Rewriting Systems

NAKAMURA Masaki

Kanazawa Univ. (~ 2011 Mar.)

→ Toyama Prefectural Univ. (2011 Apr.~)

Topic

- Give a way to construct **CafeOBJ specifications** whose corresponding TRSs are canonical (**terminating** and confluent)
  - There are lots of studies for termination and confluence in term rewriting area
  - Try to apply them to CafeOBJ specifications
  - A kind of survey of termination methods for CafeOBJ
- Throughout the presentation, only simple specifications are treated:
  - No conditional equations
  - No operator attributes

CafeOBJ and Term Rewriting System

- The reduction command in CafeOBJ is implemented based on the term rewriting system (TRS)
- For proving an equation in CafeOBJ, decompose it (make a proof score (proof passages)) until those leaves can be proved by the reduction command (= TRS)

$$\dfrac{\ddots \quad \vdots \quad ⋰}{\mathrm{NAT}+ \vdash (\forall y)0 + sy = s(0 + y)} \qquad \dfrac{\ddots \quad \vdots \quad ⋰}{\mathrm{NAT}+ \cup \{(\forall y)a + sy = s(a + y)\} \vdash (\forall y)sa + sy = s(sa + y)}$$

$$\dfrac{\mathrm{NAT}+ \vdash (\forall y)0 + sy = s(0 + y),\ \mathrm{NAT}+ \vdash (\forall y)s0 + sy = s(s0 + y),\ \mathrm{NAT}+ \vdash (\forall y)ss0 + sy = s(ss0 + y),\ \ldots}{\mathrm{NAT}+ \vdash (\forall x)(\forall y)x + sy = s(x + y)}$$

Term rewriting system

- The term rewriting system (TRS) gives us an efficient way to prove equations by regarding an equation as a left-to-right rewrite rule

```
eq N + 0 = N .
eq M + s N = s (M + N) .
```

$\Rightarrow$

```
N + 0 --> N .
M + s N --> s (M + N) .
```

`s 0 + s s 0` $=_E$ `s s 0 + s 0` ?

`s 0 + s s 0` $\rightarrow$ `s (s 0 + s 0)` $\rightarrow$ `s s (s 0 + 0)` $\rightarrow$ `s s s 0`

`s s 0 + s 0` $\rightarrow$ `s (s s 0 + 0)` $\rightarrow$ `s s s 0`

Redex and Rewriting

- A redex is an instance of the LHS of an equation
  - [Convention] A variable is written in a capital letter in this presentation

```
eq M + s N = s (M + N)
```

Redexs: `0 + s 0`   `s s 0 + s s 0`   `s X + s s (Y + s Z)`

- (1-step) Rewriting is a replacement of a redex with the corresponding instance of the RHS

```
s (0 + s s 0) --> s s (0 + s 0)
```

with

```
M <- 0    N <- s 0
```

Reduction and Normal form

- Reduction is repetition of rewriting until it cannot
- A reduced term is called a normal form
  - A term is a normal form ⇔ It has no redex

    (for unconditional TRSs)

```
s (0 + s 0) --> s s (0 + 0) --> s s 0
```

```
eq N + 0 = N .
eq M + s N = s (M + N) .
```

Variable conditions for TRS

- Rewrite rules should satisfy the following variables conditions
  1. Any LHS should not be a variable
     - E.g. by N = N + 0, reduction does not terminate

```
s 0 --> s 0 + 0 --> (s 0 + 0) + 0 --> ...
```

  2. Any RHS should not have extra variables, which are variables not included in the LHS
     - E.g. by 0 = N * 0, a redex can be rewritten into infinitely many terms (not finitely-branching)

```
   0 * 0   <--         --> s s 0 * 0 ...
                 0
 s 0 * 0   <--         --> s^n 0 * 0 ...
```

Bad equations ignored

- The reduction command in CafeOBJ ignores equations with extra-variables
  - They can be used in the apply command

```
CafeOBJ> mod* TEST{. . .
 eq N = N + 0 .
 eq 0 = N * 0 .
}
CafeOBJ> select TEST
TEST> red 0 .
-- reduce in TEST : (0):Nat
(0):Nat
(0.000 sec for parse, 0 rewrites(0.000 sec), 0 matches)
TEST>
```

TRS may not be complete

- In general, TRS achieves only a partial equational reasoning because TRS may not terminate or does not apply equations in right-to-left direction
  - a = b may not be proved, when {a = a, a = b}

$$
\begin{tikzcd}
a \arrow[dr] & =?= & b & & \\
& a \arrow[r] & a \arrow[r] & a \arrow[r] & \cdots
\end{tikzcd}
$$

  - b = c is not reduced to true, when {a = b, a = c}

$$
\begin{tikzcd}
& a \arrow[dl] \arrow[dr] & \\
b & =?= & c
\end{tikzcd}
$$

Termination and confluence

- To obtain a complete equational reasoning, the following properties are important:
  - **[Def]** A TRS is terminating if the length of every rewriting sequence is finite

$$t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots\cdots$$

  - **[Def]** A TRS is confluent if all terms obtained by rewriting from one ancestor term can be reduced into a common descendant term

$$\begin{tikzcd}
 & t_0 \arrow[dl, dotted] \arrow[dr, dotted] & \\
t_1 \arrow[dr, dotted] & & t_2 \arrow[dl, dotted] \\
 & t_3 &
\end{tikzcd}$$

Canonical TRS

- A TRS is called canonical when it is both terminating and confluent
- For a canonical TRS, the normal form of a given term is unique, so
- **[Thm]** For a canonical TRS E, every equation deducible by given equations (axioms) can be proved by the reduction command, i.e.,

$$t =_{E} t' \quad \Longleftrightarrow \quad \texttt{red } t \texttt{ = } t' \text{ returns } \texttt{true}$$

Termination and confluence undecidalbe

- In general, both termination and confluence properties are undecidable, i.e. there is no algorithm to solve the problem: Is a given TRS terminating (or confluent)?
  - It is known that confluence is decidable for a **terminating** TRS
  - **Termination** guarantees that we can compute a normal form in finite time
- Thus, constructing terminating TRS is the top priority for our purpose

Constructing terminating TRS

- Several termination methods have been proposed
- The recursive path ordering (RPO) is one of the most classical termination methods
  - A well-founded order on terms obtained from a given precedence order $\triangleright$ on operation symbols
    - E.g. `*` $\triangleright$ `+` $\triangleright$ `s`, `0`
- **[Th]** If for every rewrite rule, the left-hand side is greater than the right-hand side by RPO, then the TRS is terminating

Recursive Path Ordering

- [Def] $s = f(s_1,\ldots,s_m) >_{rpo} t$ if
  1. $s_i \geq_{rpo} t$ for some i, or
  2. $t = g(t_1,\ldots,t_n)$, $f \rhd g$ and $s >_{rpo} t_j$ for every j, or
  3. $t = g(t_1,\ldots,t_n)$, $f = g$ and $[s_1,\ldots,s_m] >_{rpo}^{mul} [t_1,\ldots,t_n]$

```
eq N +   0 = N .
eq M + s N = s (M + N)
```

- s N $>_{\text{rpo}}$ N (From 1)
  - [M, s N] $>^{\text{mul}}$ [M, N]
- M + s N $>_{\text{rpo}}$ M + N (From 3)
- M + s N $>_{\text{rpo}}$ s(M + N) (From 2 with + $\rhd$ s)

Constructing RPO-terminating TRS

- [Def] A root symbol of the left-hand side of some rewrite rule is called **a defined symbol** (**D**)
- Construct a TRS as follows:
  - Every occurrence $\mathtt{g(r_1,...r_n)}$ of a defined symbol $\mathtt{g}$ in every right-hand side should satisfy
    - Every $\mathtt{r_i}$ is a subterm of some argument $\mathtt{l_j}$ of the left-hand side $\mathtt{f(l_1,...l_m)}$
    - At least one $\mathtt{r_i}$ is a strict subterm of some $\mathtt{l_j}$
- Then, the TRS can be proved terminating by RPO with the precedence order defined as **D** $\triangleright$ **C** (Constructor (Non-defined) symbols)

Examples

```
eq N +   0 = N .
eq M + s N = s (M + N)
```

$+ \triangleright$ s, 0

```
eq   0 -   N = 0 .
eq   M -   0 = M .
eq s M - s N = (M - N)
```

$- \triangleright$ s, 0

```
eq N *   0 = 0 .
eq M * s N = M + (M * N)
```

$* \triangleright$ +, s, 0

```
eq even(0) = true .
eq odd(0) = false .
eq even(s N) = odd(N) .
eq odd(s N) = even(N) .
```

Even,odd $\triangleright$ 0,true,false

- Every **single** module (TRS) above can be proved terminating by RPO with the precedence
- How about a combination of them ... ?

Modularity

- **[Def]** A property P is modular for TRSs if for all TRSs R and R' having P, their combination R U R' also has P
  - **Question**: Is termination modular?
  - **Answer**: No
    - Even if R and R' has no sharing operation symbols, termination is not modular

Toyama's famous counterexample

```
eq f(0,1,X) = f(X,X,X)) .
```

```
eq g(X, Y) = X .
eq g(X, Y) = Y .
```

f(0,1,g(0,1)) → f(g(0,1),g(0,1),g(0,1))

→ f(0,g(0,1),g(0,1)) → f(0,1,g(0,1))

C_E-termination

- **[Def]** A TRS R is C_E-terminating if R U C_E is terminating, where C_E={cons(x,y)→x, cons(x,y)→y}
- **[Th]** C_E-termination is modular for disjoint TRSs

```
eq f(0,1,X) = f(X,X,X)) .
```

```
eq g(X, Y) = X .
eq g(X, Y) = Y .
```

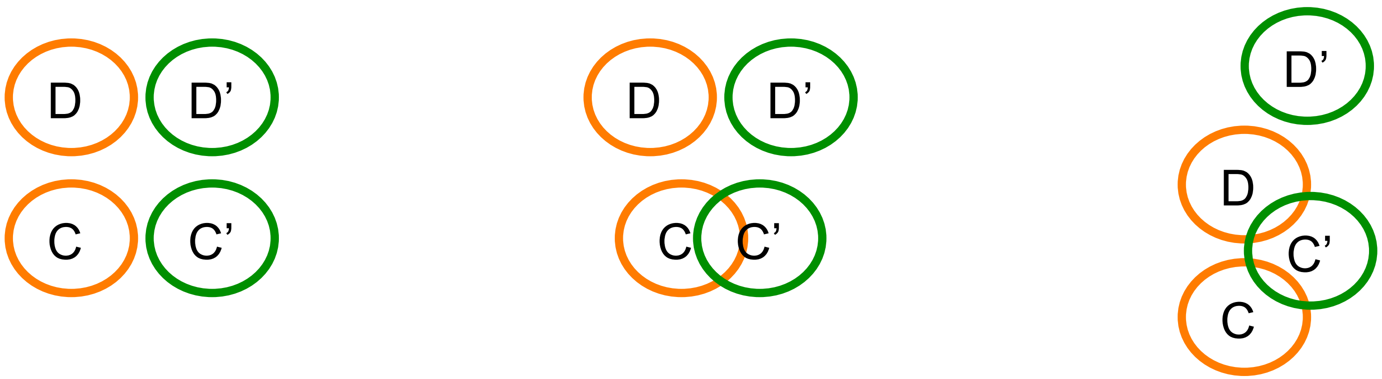
Simple Termination ⇒ C_E- Termination ⇒ Termination

RPO

- Disjoint union is too strong for CafeOBJ specifications
  - Importing and imported modules usually share operation symbols

Kinds of Combinations

- There are different kinds of combinations
  1. R and R’ are disjoint if they do not have share operation symbols
  2. R and R’ are constructor-sharing if they share at most constructors
  3. A hierarchical combination of the base system R and the extension R’ allows defined symbols of R to occur as constructors in R’ (not vice versa)



Examples of combinations

- R_+ and $R_@$ are disjoint
- R_+ and R_- are constructor-sharing
  - s and 0 are shared
- R_* is an extension of the base system R_+ (hierarchical combination)
  - + is a constructor in R_*

```
eq N +   0 = N .
eq M + s N = s (M + N)
```

```
eq nil @ YS = YS .
eq (X : XS) @ YS = X : (XS @ YS).
```

```
eq   0 -   N = 0 .
eq   M -   0 = M .
eq s M - s N = (M - N)
```

```
eq N *   0 = 0 .
eq M * s N = M + (M * N)
```

C_E-termination and constructor-sharing

- **[Def]** A TRS is finitely-branching if for all terms t, the set { t' | t $\rightarrow$ t' } of one-step reducts of t is finite
- [**Th**] C_e-termination is modular for finitely-branching constructor-sharing TRSs

- Trivially, if the number of equations (rewrite rules) is finite (= finite TRS), the TRS is finitely-branching
- [**Cr**] C_e-termination is modular for finite constructor-sharing TRSs

C_E-termination and hierarchical combinations

- In general, C_E-termination is not modular for hierarchical combinations

```
eq a = b .
```

```
eq f(b) = f(a) .
```

```
eq N +   0 = N .
eq M + s N = s (M + N)
```

```
eq N *   0 = 0 .
eq M * s N = M + (M * N)
```

- What is the difference between the upper one and the lower one?
  - The occurrence of the defined symbols (**a** and **+**) of the base system in the R.H.S. of the extension

Restricted proper extension

- **[Def]** R' is a proper extension of R if functions depending on R are never called **within** a recursive call of R'
  - f depends on R ⇔ f(...) = C[g(...)] exists for some g in D
- **[Def]** R' is a restricted proper extension of R if it is a proper extension of R such that no L.H.S of R' contains defined symbols strictly below its root
- **[Th]** C_E-termination is modular for finite restricted proper extensions
  - For more precise definitions, see the reference [Ohlebusch 2002]

```
eq a = b .
```

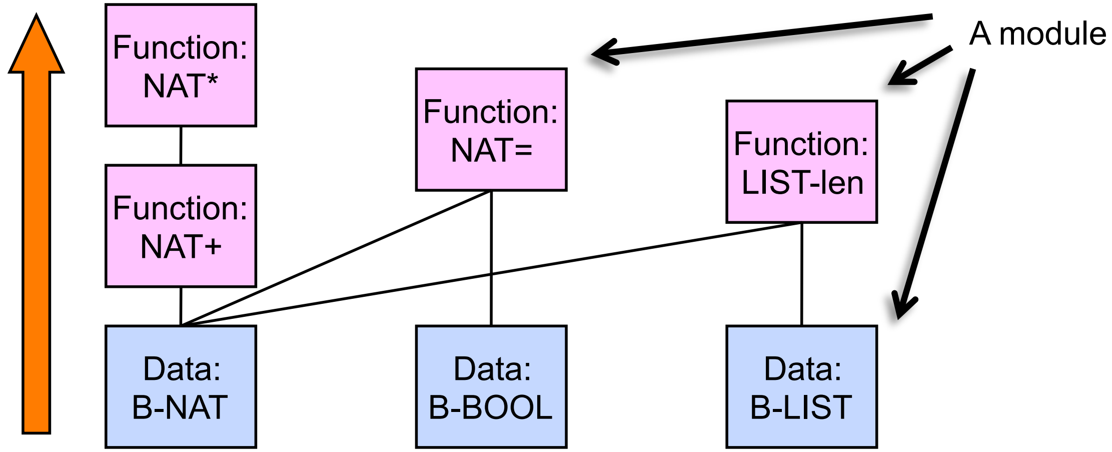
```
eq f(b) = f(a) .
```

```
eq N +   0 = N .
eq M + s N = s (M + N)
```

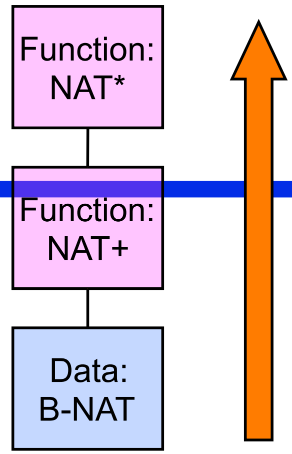
```
eq N *   0 = 0 .
eq M * s N = M + (M * N)
```

Apply modularity results to CafeOBJ spec.

- To describe a specification of an abstract data type
  - Describe a module for constructors
  - Give a partial order on functions to be defined
    - Like `power() > _*_ > _+_`
  - Describe a module for each function on the data type (one module for one function)



Constructors

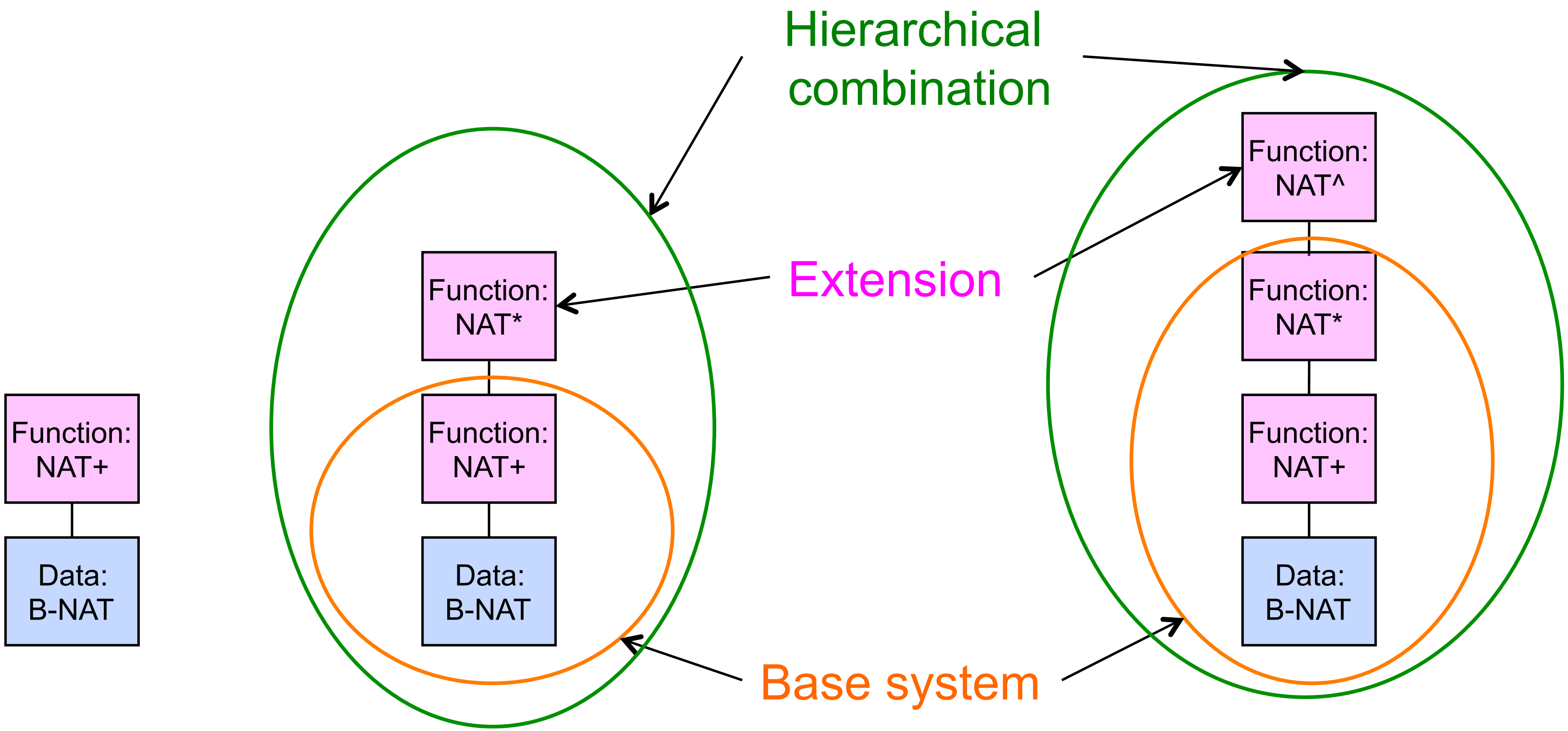


- Equations in each module for functions should satisfy the following conditions:
  - Every occurrence $\mathtt{g(r_1,...r_n)}$ of a defined symbol in every R.H.S. should satisfy that
    - Every $\mathtt{r_i}$ is a subterm of some argument $\mathtt{l_j}$ of the L.H.S. $\mathtt{f(l_1,...l_m)}$
    - At least one $\mathtt{r_i}$ is a strict subterm of some $\mathtt{l_j}$

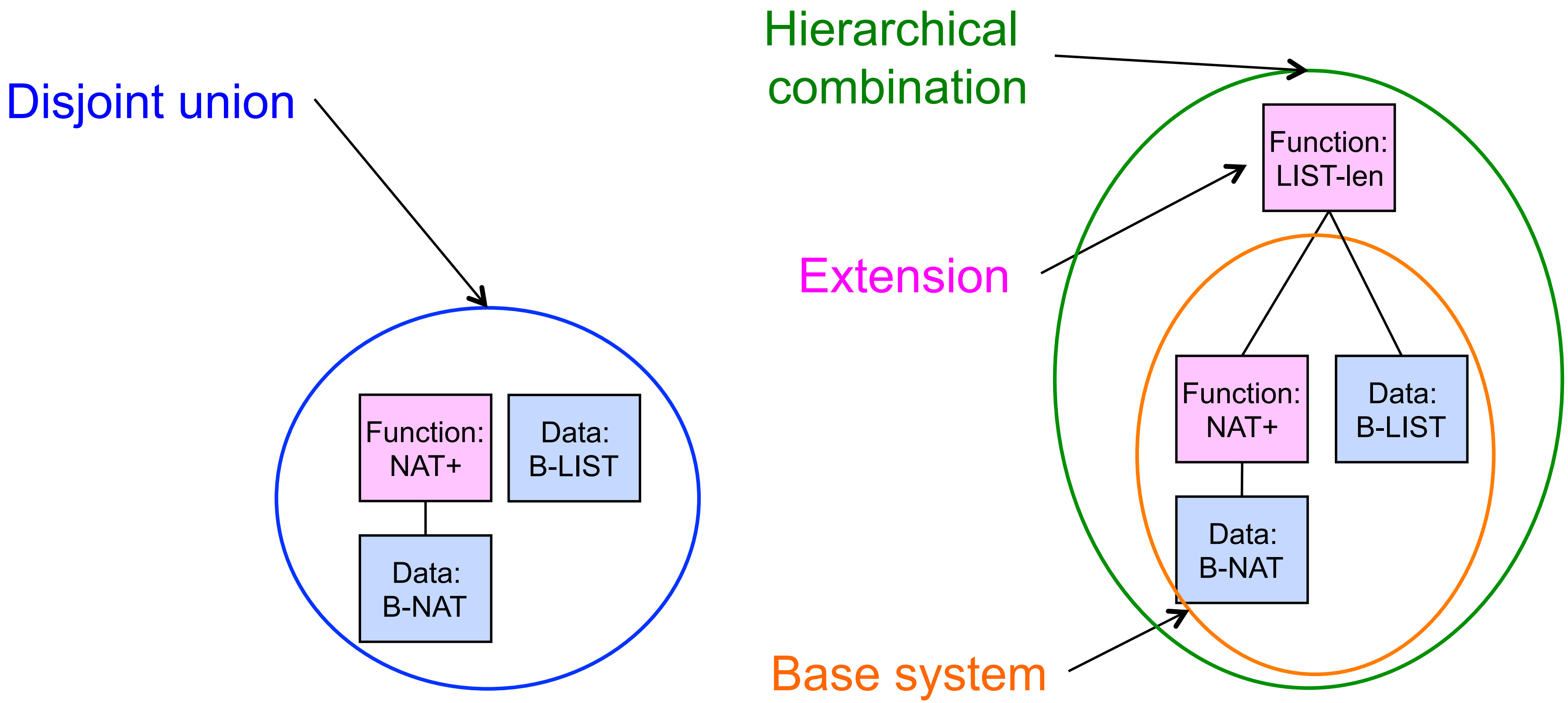
For RPO

  - Each argument $\mathtt{l_i}$ is a constructor term for every `eq` $\mathtt{f(l_1, l_2, ..., l_n) = r}$

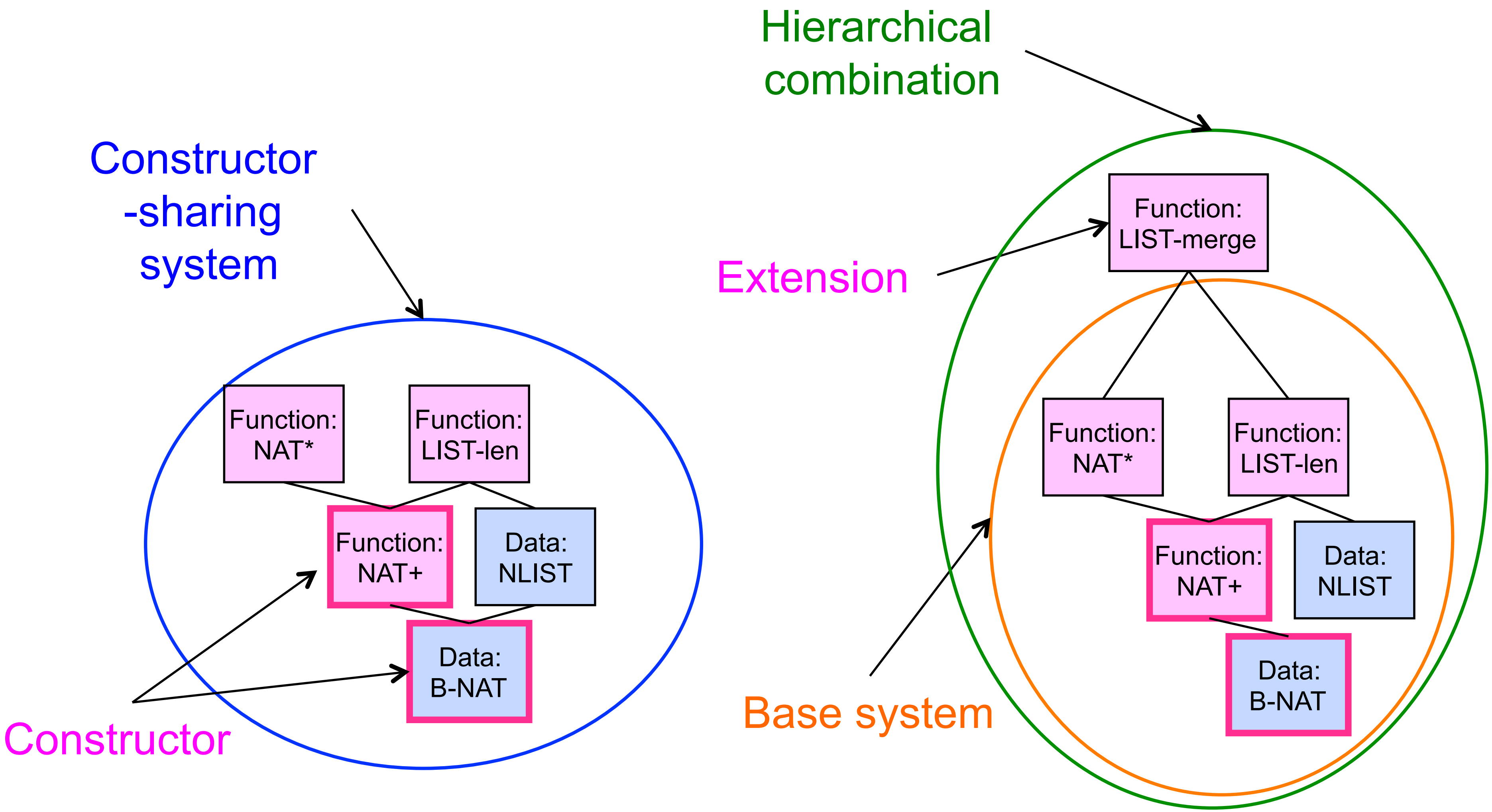
Constructing terminating specification



Constructing terminating specification



Constructing terminating specification



Out of scope

- There are CafeOBJ specifications which do not satisfy the above conditions of the hierarchical design:
  - Module **INT** with **s p = X** and **p s X = X**
    - Since **s** is **D**, **eq X + s Y = s(X + Y)** has **D** in the arguments L.H.S => not restricted proper ext.
  - Built-in modules may include
    - infinitely many constants 0,1,2,3,...
    - infinitely many 0+1=1, 1+1=2, ...
  - Proof scores
    - I.H.: eq n + m = m + n
    - lemma: eq X * (Y + Z) = X * Y + X * Z

Future work

- Extend the theorems of hierarchical combinations to those covering CafeOBJ specifications
  - INT, built-in modules, proof score
  - Operator attributes: AC-TRS
  - Conditional equations:
    - 1-CTRS (conditions have no extra variable)
    - Normal CTRS (condition is interpreted as reachability to the constant **true**)

    ceq l = r if cond. ➔ l → r if cond →* true