Implementation of An Operational Semantics for MSO Logic over Trees

by
Ari Saptawijaya

Master's Thesis
International Master Programme
Computational Logic

Revised Version

Chair for Foundations of Programming
Institute of Theoretical Computer Science
Department of Computer Science
Dresden University of Technology

Supervising Professor: Prof. Dr.-Ing. habil. Heiko Vogler
Supervisor: Dipl.-Inf. Enrico Bormann

Acknowledgments

First of all, I would like to thank Enrico Bormann as a wonderful supervisor, for his patience and great support during the whole preparation of my thesis. I am very glad for having many fruitful discussions with him and also constructive suggestions from him, which I believe have improved a lot the way I prepared my thesis. I also would like to thank Prof. Heiko Vogler and Dr. Armin Kühnemann who have introduced me to this interesting field from the very beginning I joined the International Master Programme in Computational Logic. I am indebted to them for many valuable suggestions during my study here. It has also been a pleasure for me to work in the Chair for Foundations of Programming, where I really could enjoy a friendly and welcoming atmosphere.

I would also like to thank Endrawaty and Defa Trio for helping me in the proof-reading of my report.

I am glad to join the International Master Programme in Computational Logic, which I found a great programme, thus I thank the people who have made it so. I am very grateful for having a financial support from the International Quality Network "Rational Mobile Agents and Systems of Agents", that enabled me to come to Dresden and studied in this programme. Many thanks to them, who have made the decision for sponsoring my stay and study in Dresden.

Last, but not least, I want to thank my parents for always being there.

Contents

Chapter 1

Introduction

In system development, there is variety of formal languages which are used by developers to develop the systems. For example, to deliver a software product some formal languages might be involved in the development, e.g. specification languages and programming languages. The availability of an editor which is based on such a formal language would be very helpful for developers to support their works in developing the system.

In [2], a system that automatically generates an editor based on a formal language was designed and implemented. Such an editor is also called a language-based editor. A language-based editor offers some convenient features for the users. For instance, a program editor may provide a general template of the language for which the editor is generated and also allows structural editing. It is also able to perform an immediate analysis, translation, and error reporting while an object is being edited. Errors are detected as soon as they occur during editing of the object.

To generate such an editor for a particular formal language L, the specification of L is required which serves as the input for the generator. The specification of L consists of two parts, viz. the context-free and the context-sensitive part of L. As the specification language, we use the so-called *DSL* (*Dresden Specification Language*) [3]. This specification language combines the paradigms of attribute grammars, functional programming, and formal logics, such that it allows a comfortable specification of the syntax of a formal language.

We give an example of a programming language specification. Consider an imperative programming language, which we call *Simple*. The programming language *Simple* mainly consists of three parts, viz. the part for the name of the edited program, the declaration part, and the statement part. It recognizes two data types, i.e. integer and boolean data types, and three statement constructs, i.e. the assignment of a variable, the if-then-else statement, and the while-loop statement. Expressions can be built up by variables, constants, and two operators, i.e. addition and test of equality. A context-sensitive restriction for this language is that a variable must be first declared before it is used. An example of program which can be written in the programming language *Simple* is shown in figure 1.1.

A violation of the context-sensitive restriction is immediately detected and informed to the user of the editor by giving an appropriate error message. For instance, if a user of the editor for the language *Simple* want to use a variable

```
program Count;
var
  flag : Bool;
  sum : Int
begin
  flag = True;
  sum = 0;
  while (flag == True)
  do
    if (sum == 10)
    then
      flag := False
    else
      sum := (sum+1)
end
```

Figure 1.1: An example of program written in the programming language *Simple*

```
program Count;
var
  flag : Bool;
  sum : Int
begin
  flag := True;
  num {Not Declared}:= <exp>
end
```

Figure 1.2: A violation of the context-sensitive restriction for the language *Simple*

num which is not declared, the editor will response by giving the message `{Not Declared}` as shown in figure 1.2.

To specify the language *Simple* in DSL, its abstract syntax is used. This abstract syntax is obtained by abstracting the context-free grammar from its terminal symbols and giving each production of the context-free grammar a unique name. For instance, consider the following production of the context-free grammar of *Simple*:

$$\texttt{Program} \rightarrow \textit{program}\ \texttt{Ident}\ ;\textit{var}\ \texttt{DList}\ \textit{begin}\ \texttt{SList}\ \textit{end}$$

By abstracting from its terminal symbols and giving this production the name `PROG`, we get the following part of the abstract syntax:

```
Program = PROG Ident DList SList
```

In DSL, the abstract syntax of L is defined within a construct, viz. *data block*, where the abstract syntax is encapsulated by the keywords `begindata` and `enddata`. For example, the language *Simple* can be specified by the following abstract syntax:

```
begindata Simple_Data
  data Program  = PROG Ident DList SList
  data DList    = DSINGLE Decl
                | DPAIR Decl DList
  data Decl     = DECL Ident TypExp
  data TypExp   = EMPTYTYP
                | INTTYP
                | BOOLTYP
  data Ident    = IDENTNULL
                | IDENT String
  data SList    = SSINGLE Stmt
                | SPAIR Stmt SList
  data Stmt     = EMPTYSTMT
                | ASSIGN Ident Exp
                | IF Exp Stmt Stmt
                | WHILE Exp Stmt
  data Exp      = EMPTYEXP
                | INTCONST Int
                | BOOLCONST Bool
                | ID Identifier
                | EQUAL Exp Exp
                | ADD Exp Exp
enddata
```

This abstract syntax can be seen as the rules of a regular tree grammar which is related to the context-free grammar of the language *Simple*, where `PROG` (and also other constructors of every rule) is a terminal symbol of the related regular tree grammar. In this thesis, we shall consider a *sorted* regular tree grammar, where both terminal and nonterminal symbols are equipped with sorts. In particular, the sort of a terminal symbol encodes a rule of the regular tree grammar. This is possible, since a terminal symbol is uniquely associated to every rule in the regular tree grammar. For example, the terminal symbol `PROG` in our example is provided with the following sort:

```
(Ident DList SList,Program)
```

which encodes the rule:

```
    Program = PROG Ident DList SList
```

By specifying the language using its abstract syntax, an object edited by the language-based editor is internally represented by an abstract syntax tree. Moreover, in order to be able to specify the context-sensitive restriction, an attribute grammar is used in the specification. By using an attribute grammar, the nodes of an abstract syntax tree are equipped with attributes. For an example, every node labeled by `IDENT` is equipped with the attribute *name*, whose value is the name of the corresponding identifier. Moreover, we also give a sort to every node in the tree. The sort of a node is determined by the last component of the sort of the node's label. For example, the node labeled by `PROG` has the sort `Program`, since `PROG` has the sort `(Ident DList SList,Program)`. We shall define this more formally in this thesis.

To specify the context-sensitive part of a language, one can use a logical formalism, which enables the specifier of an editor to specify the context-sensitive part logically and globally. Since the edited object is represented internally as a tree, the *monadic second order (MSO) logic over trees* seems to be an appropriate logical formalism for this purpose, since this logical formalism is able to define relations over nodes (or set of nodes). But, in order to be able to specify the context-sensitive syntax of a language, it is necessary to relate this logical formalism with the attribute grammar formalism.

In specifying the context-sensitive syntax of a language, the MSO formulas are used to define the so-called *node properties* with respect to the considered abstract syntax tree. A node property is a property which is expressed by an MSO formula with one free node variable only. By node variable, we mean a variable which is used to denote a node in a tree. Let us consider again the context-sensitive restriction of the programming language *Simple*, viz. a variable must be declared before it is used. Consider also the program given in figure 1.2, where we try to use the variable `num` in the statement part. To check whether the context-sensitive restriction is violated, internally a test should be performed to see whether there exists a node labeled by `IDENT` in the declaration part where the attribute *name* of this node has the value `num` (which is not the case for our example). This test can be specified by using an MSO formula that defines a node property. This property can be asked to every node labeled by `IDENT` in the statement part, whose value of the attribute *name* needs to be checked in the declaration part (whether there is also a node labeled by `IDENT` with the same value of attribute *name*).

Having this possibility of using the MSO logic over trees to specify the context-sensitive syntax of a language, there is a need for this logic to be integrated into the concept of the generator system. Therefore, an operational semantics for this logic based on attribute grammars has to be defined. To achieve this goal, we give the solution of several subtasks in this thesis:

- We define a variant of MSO logic over trees by defining additional atomic formulas which are appropriate for specifying the context-sensitive syntax of a formal language. In particular, we define a so-called *attribute formula* as an atomic formula. With this formula we are able to relate the attribute grammar and the MSO logic formalism. For example, by using this formula we are able to compare the attribute values of some nodes as we did in the previous example, where we compare the values of the attribute *name* of the nodes labeled by `IDENT` in the statement and in the declaration part. Hence, by introducing this formula into MSO logic, we are not only able to specify context-free phenomena, but also context-sensitive phenomena. We shall call this variant of logic as *MSO* logic.* To be able to relate these two formalisms, we introduce formally an interface between them.

- To define an operational semantics for MSO* logic, we generalize the known transformation of an MSO formula into a tree automaton (which recognizes the tree language of this formula) [5, 13] for MSO* logic. By adding the attribute formula into the logic, the generalized transformation of this formula yields a tree automaton with *infinite* number of states. Therefore, in this thesis we also propose a definition of a tree automaton with infinite number of states, which fits our needs.

- We propose a representation of tree automata which can deal with infinite number of states. To represent an automaton, in particular we give the representation of transitions. This representation is needed to construct the transition functions of a tree automaton. We need to represent the transitions, since to produce the transition functions, an inductive construction (on the structure of the formula) is applied. Such inductive construction will often produce intermediate results. Thus, in our representation of transitions and the inductive construction (which is based on our representation) we attempt to minimize the number of transitions in several ways, i.e.

  - by having assumption, that our automaton only runs on special trees, viz. *well-marked trees*,
  - by considering the sort information of symbols which label the nodes of a tree,
  - by having a representation of input symbols which represents input symbols of transitions with the same behavior,
  - by considering only reachable states in the inductive construction.

  At the end of the inductive construction, the final representation of transitions are produced. From this representation, we can derive the transition functions for which they are represented.

  The accepting functions are also defined inductively on the structure of the formula, but instead of having a representation for the set of final states, we prefer to produce the definition of the accepting functions of a formula and its subformulas directly. By keeping the structure of the state (which corresponds to the structure of the formula), these definitions of accepting functions can be applied inductively on the structure of the state, which is reached after the automaton has run on a tree, to decide whether the tree is accepted or not.

  Since we deal with tree automata having infinite number of states, we have to find an appropriate representation of states. In this thesis, we propose a symbolic representation to be able to represent such infinite number of states. When we derive a transition function from its representation, this symbolic representation can be seen as a parameter or placeholder which will be substituted by a concrete value when we let the automaton run on a tree.

  There is a subtle issue in choosing the kinds of symbolic representation we need. In fact, a symbolic representation is introduced for representing a node in a tree, which is collected in a state of an attribute formula as the value of node variable occurring in the attribute formula. But, since we can have a formula with quantifier over a node variable and an attribute formula may occur as its subformula, where one of the node variables in this attribute formula is the quantified variable, then a symbolic representation of only one node is not enough. The reason is that if we quantify over a node variable, then this means we make some guessing of nodes for the possible value of the quantified node variable. In particular, if the quantified variable occurs in an attribute formula, then there is a need to have a symbolic representation that can represent this set of guessed nodes. Therefore, in this thesis we introduce two kinds of symbolic representation, viz. a symbolic representation of a node and a symbolic representation of a set of nodes.

The choice of a suitable representation for tree automata is important, since it is already known that the complexity of the transformation has a *non-elementary* lower bound of state space: $\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}n$, where n is the number of quantifiers [10, 12]. Some similar works in dealing with an appropriate representation of automaton can be found in [1, 7, 8, 9, 11]. In contrast to our work, none of these works have considered the relation between the MSO logic and the attribute grammar formalism. Moreover, these works only deal with string or binary trees and the tree automata have only a finite number of states.

- We implement the inductive construction of tree automata based on our representation. For the compatibility to the existing implementation of the generator system, the implementation is also realized in the functional programming language *Haskell*. The implementation produces for a given set of MSO* formulas (which specify the context-sensitive part of a language), a Haskell code which contains the transition functions and the accepting functions of those formulas. The transition functions of a formula are derived by unfolding our representation of transitions resulting from the final step of inductive construction of the formula, whereas the accepting function is derived inductively from the structure of the formula.

  In general, the current implementation has already included all the construction needed to produce the transition functions and the accepting function of a formula. But, there are still some works which have not been implemented yet, viz. the implementation for the refinement of the power set construction, in case it is used to determinize the representation of nondeterministic transitions which come from a quantified formula in which an attribute formula is involved and one of the node variable in this attribute formula is the quantified variable. We also have not considered this case in the implementation of the accepting function. Nevertheless, in this thesis we give a proposal of solution, which can be implemented.

We organize the presentation of this thesis as the following:

- Chapter 2 contains the necessary notations and notions we use throughout the report.
- Chapter 3 discusses the context-free grammar and the sorted regular tree grammar which is related to a context-free grammar. It also gives the definition of abstract syntax trees formally.
- Chapter 4 gives the necessary interface that enables the combination of MSO* logic and the attribute grammar formalism.
- Chapter 5 discusses our proposal of a definition for tree automata with infinite number of states, which we use to define the operational semantics of the MSO* logic.

- Chapter 6 presents the MSO* logic by its syntax and also its semantics. We give both the model theoretical and the operational semantics of this logic.
- Chapter 7 discusses our proposal of the inductive construction of tree automata. We show how we represent the extended ranked alphabet, the states, and the transitions (which is used eventually to produce the transition functions). By using this representation, we define then the cross product construction, the projection construction, and the power set construction. In this chapter, we also propose an additional operation to combine some representation of transitions (resulting from each of the above mentioned construction) which have the same behavior.
- Chapter 8 discusses how we implement our construction to produce the transition functions and the accepting functions of a given set of MSO* formulas as a Haskell code.
- Chapter 9 concludes our work and proposes some future works.

Chapter 2

Preliminaries

In this chapter we recall some basic notions and notations we use throughout the report. We give the definition of sorted sets, sorted ranked alphabets, graphs, sorted terms, sorted trees, and term-trees relatedness.

2.1 General Notations

We denote the set of non-negative integers by $\mathbb{N}$ and the set of positive integers by $\mathbb{N}_+$. For every $m \in \mathbb{N}$, we use $[m]$ to denote the set $\{1, \ldots, m\}$. We use *Bool* to denote the set $\{True, False\}$ of truth values.

For every $k \in \mathbb{N}$ and for k sets $M_1, \ldots, M_k$, the cartesian product of $M_1, \ldots, M_k$ is denoted by $M_1 \times \ldots \times M_k = \{(m_1, \ldots, m_k) | m_i \in M_i, i \in [k]\}$. Every subset $R \subseteq M_1 \times \ldots \times M_k$ is called k-ary relation over $M_1, \ldots, M_k$. For three sets M_1, M_2, and M_3 and two binary relations $R_{1,2} \subseteq M_1 \times M_2$ and $R_{2,3} \subseteq M_2 \times M_3$, the composition of $R_{1,2}$ and $R_{2,3}$ is denoted by $R_{1,2} \circ R_{2,3} = \{(e_1, e_3) \mid (e_1, e_2) \in R_{1,2}$ and $(e_2, e_3) \in R_{2,3}\}$. For every set M and for every binary relation $R \subseteq M \times M$ the transitive closure of R is denoted by R^+. The reflexive and transitive closure of R is denoted by R^*.

For every $k \in \mathbb{N}$ and for $k+1$ sets $M_0, M_1, \ldots, M_k$, a k-ary function f over $M_1, \ldots, M_k$ into M_0 is denoted by $f : M_1 \times \ldots \times M_k \to M_0$. We denote the cardinality of a set M by $card(M)$. We use $max(M)$ to define the maximum of the elements of $M \subseteq \mathbb{N}$ if M is a non-empty set and 0, otherwise.

Let Σ be a finite set of symbols. The set of all finite words over Σ is denoted by Σ^*. The empty word is denoted by ε.

2.2 Sorted Sets and Sorted Ranked Alphabets

Definition 1 (Sorted Set). Let S be a non-empty finite set of sorts. An *S-sorted set* is a pair $(\Sigma, sort)$, where Σ is a set of symbols and $sort : \Sigma \to S$ is a mapping which maps every symbol $\sigma \in \Sigma$ to a sort $s \in S$.

A symbol $\sigma \in \Sigma$ with $sort(\sigma) = s$ is also denoted by σ^s. For every sort $s \in S$, Σ^s denotes the subset of Σ that contains all symbols σ^s. If the mapping *sort* of an arbitrary sorted set $(\Sigma, sort)$ is known from the context, then $(\Sigma, sort)$ is abbreviated by Σ. □

Definition 2 (Sorted Ranked Alphabet). Let S be a non-empty finite set of sorts. An *S-sorted ranked alphabet* is a pair $(\Sigma, rank)$, where Σ is a finite $(S^* \times S)$-sorted set and $rank : \Sigma \to \mathbb{N}$ is a mapping which maps every symbol $\sigma \in \Sigma^{(s_1 \cdots s_k, s_0)}$ with $k \in \mathbb{N}$ and $s_0, s_1, \ldots, s_k \in S$ to the rank k.

For every rank $k \in \mathbb{N}$, Σ_k denotes the subset of Σ that contains all symbols σ with the rank k. If the mapping *rank* of an arbitrary sorted ranked alphabet $(\Sigma, rank)$ is known from the context, then $(\Sigma, rank)$ is abbreviated by Σ.

We use $Rank_\Sigma$ to denote the set of ranks of elements in Σ, i.e. $Rank_\Sigma = \{k \in \mathbb{N} \mid k = rank(\sigma), \sigma \in \Sigma\}$. We use $Rank_{\Sigma,+}$ to denote the set of positive ranks of elements in Σ, i.e. $Rank_{\Sigma,+} = Rank_\Sigma \setminus \{0\}$.

We define the function $sort_{res} : \Sigma \to S$, such that for every $\sigma \in \Sigma^{(s_1 \cdots s_k, s_0)}$ with $k \in \mathbb{N}$ and $s_0, s_1, \ldots, s_k \in S$, $sort_{res}(\sigma) = s_0$. In the other words, we say s_0 is the *result sort* of σ. We also define the function $sort_{in_i} : \Sigma \to S$, such that for every $\sigma \in \Sigma^{(s_1 \cdots s_k, s_0)}$ with $k \in \mathbb{N}_+$, $i \in [k]$, and $s_0, s_1, \ldots, s_k \in S$, $sort_{in_i}(\sigma) = s_i$. In the other words, we say s_i is the *i-th input sort* of σ. The maximum rank of an arbitrary sorted ranked alphabet Σ is denoted by $max\_rank(\Sigma) = max(Rank_\Sigma)$. □

2.3 Graphs

Definition 3 (Directed Labeled Graph). Let Σ be a set of symbols and Γ be a set of symbols as well. A *directed labeled graph over Σ and Γ* is a tuple (V, E, lab), where V is finite *set of nodes*, $E \subseteq V \times \Gamma \times V$ is a finite *set of labeled edges*, and $lab : V \to \Sigma$ is the *node-labeling function.*

For an edge $(v, i, w) \in E$, the node $v \in V$ has (v, i, w) as its *outgoing edge* and the node $w \in V$ has (v, i, w) as its *incoming edge.* □

The following notations will be used throughout the report for directed labeled graphs. For a given graph g, the set of its nodes, the set of its edges, and its node-labeling function are denoted by V_g, E_g, and lab_g, respectively.

2.4 Sorted Terms and Sorted Trees

Definition 4 (Sorted Term). Let S be a non-empty finite set of sorts, Σ be an S-sorted ranked alphabet, and Y be an S-sorted set. The *set of S-sorted terms over Σ indexed by Y*, denoted by $T_\Sigma(Y)$, is the smallest S-sorted set $\bigcup_{s \in S} T^s$ such that:

1. for every $s \in S$, $Y^s \subseteq T^s$,
2. for every $\sigma \in \Sigma^{(s_1 \cdots s_k, s_0)}$ with $k \geq 0$ and $s_0, \ldots, s_k \in S$, and for every $te_1 \in T^{s_1}, \ldots, te_k \in T^{s_k}$, the term $\sigma(te_1, \ldots, te_k) \in T^{s_0}$.

We abbreviate $T_\Sigma(\emptyset)$ by T_Σ. □

We define in the following S-sorted trees. We consider S-sorted trees as special directed labeled graphs.

Definition 5 (Sorted Tree). Let S be a non-empty finite set of sorts and Σ be an S-sorted ranked alphabet. An *S-sorted tree t over* Σ is a directed labeled graph t over Σ and $[max\_rank(\Sigma)]$ such that:

1. there is exactly one node without incoming edges, namely the *root* of t, denoted by $root(t)$,
2. for every node $n \in V_t$ which has incoming edges, there is exactly one sequence $n_1, \ldots, n_h$ with $h \geq 1$ and $n_h = n$ such that $(root(t), i_1, n_1) \in E_t$ and for every $j \in [h-1]$, $(n_j, i_{j+1}, n_{j+1}) \in E_t$ for some $i_1, \ldots, i_h \in [max\_rank(\Sigma)]$, where $root(t)$ is the root of t,
3. V_t is an S-sorted set such that for every node $n \in V_t$, if $lab(n) = \sigma \in \Sigma^{(s_1 \ldots s_k, s_0)}$ with $k \geq 0$ and $s_0, \ldots, s_k \in S$, then $n \in V_t^{s_0}$, and for every $i \in [k]$ there is exactly one node $n_i \in V_t^{s_i}$ such that $(n, i, n_i) \in E_t$,
4. t has the same sort as the root of t, i.e. $sort(t) = sort(root(t))$.

The set of all S-sorted trees over an S-sorted ranked alphabet Σ is denoted by $TR(\Sigma)$.

For every $t \in TR(\Sigma)$ and $n \in V_t$, n is also specified by a sequence $j_1.j_2.\ldots.j_l$ with $l \geq 0$ and $j_1, \ldots, j_l > 0$ that describes the path from the root of t to the node n, which is known as the *Dewey notation*. In particular, the root of t is specified by ε. For every $n \in V_t$, $n.j$ denotes the j-th successor node n_j of n with $(n, j, n_j) \in E_t$, in case such node n_j exists.

To use the Dewey notation for specifying a node in a tree $t \in TR(\Sigma)$, we define the partial function $ids_t : (\mathbb{N}_+ \cdot \{.\})^* \to V_t$ that maps a Dewey notation $v \in (\mathbb{N}_+ \cdot \{.\})^*$ to a node $ids_t(v) \in V_t$, such that $ids_t(v) = v$. □

Definition 6 (Term-Trees Relatedness). Let S be a non-empty finite set of sorts and Σ be an S-sorted ranked alphabet. Every S-sorted term te over Σ is *related* to the set of S-sorted trees over Σ representing the term te in a graphical form via the function $gr : T_\Sigma \to \mathcal{P}(TR(\Sigma))$ which is defined by structural induction on T_Σ as follows:

- $\forall \alpha \in \Sigma^{(\varepsilon, s_0)}$ with $s_0 \in S$:
  <u>if</u> $te = \alpha$,
  <u>then</u> $gr(te)$ is the smallest set, such that $t = (V_t, E_t, lab_t) \in gr(te)$, where:

  $$V_t = \{n^{s_0}\}, E_t = \emptyset, \text{ and } lab_t(n) = \alpha.$$

- $\forall k > 0, \sigma \in \Sigma^{(s_1 \ldots s_k, s_0)}$ with $s_0, \ldots, s_k \in S$, and $\forall te_1 \in T_\Sigma^{s_1}, \ldots, te_k \in T_\Sigma^{s_k}$, and $t_1 \in gr(te_1), \ldots, t_k \in gr(te_k)$ such that $V_{t_i} \cap V_{t_j} = \emptyset$ for every $1 \leq i < j \leq k$:
  <u>if</u> $t = \sigma(te_1, \ldots, te_k)$,
  <u>then</u> $gr(te)$ is the smallest set, such that $t = (V_t, E_t, lab_t) \in gr(te)$, where:
  - $V_t = \{n^{s_0}\} \cup \bigcup_{i \in [k]} V_{t_i}$ where $n \notin \bigcup_{i \in [k]} V_{t_i}$,
  - $E_t = \{(n, i, root(t_i)) \mid i \in [k]\} \cup \bigcup_{i \in [k]} E_{t_i}$,
  - $lab_t(n) = \sigma$ and for every $m \in V_{t_i}$ with $i \in [k]$, $lab_t(m) = lab_{t_i}(m)$.

□

Chapter 3

Context-free and Sorted Regular Tree Grammars

In this chapter we define the context-free grammars, the sorted regular tree grammars, their relatedness, and the abstract syntax trees.

3.1 Context-free Grammars

Definition 7 (Context-free Grammar). A *context-free grammar* is a tuple $G_0 = (N, \Lambda, Z, P)$, where:

- N is a finite set of *nonterminal symbols*,
- Λ is a finite set of *terminal symbols*, where $\Lambda \cap N = \emptyset$,
- $Z \in N$ is the *start symbol*,
- P is a finite set of *productions* of the form $\eta_0 \to \lambda_0 \eta_1 \lambda_1 \dots \eta_k \lambda_k$ with $k \geq 0$, $\eta_0, \dots, \eta_k \in N$, and $\lambda_0, \dots, \lambda_k \in \Lambda^*$. □

Definition 8 (Derivation Relation of Context-free Grammar). Let $G_0 = (N, \Lambda, Z, P)$ be a context-free grammar. The *derivation relation of* G_0, denoted by $\Rightarrow_{G_0}$, is the smallest binary relation $\Rightarrow \subseteq (N \cup \Lambda)^* \times (N \cup \Lambda)^*$ such that for every $\xi_1, \xi_2 \in (N \cup \Lambda)^*$, $(\xi_1, \xi_2) \in \Rightarrow$ if there exist two words $\xi', \xi'' \in (N \cup \Lambda)^*$ and a production $\eta_0 \to \lambda_0 \eta_1 \lambda_1 \dots \eta_k \lambda_k \in P$ such that $\xi_1 = \xi' \eta_0 \xi''$ and $\xi_2 = \xi' \lambda_0 \eta_1 \lambda_1 \dots \eta_k \lambda_k \xi''$. Instead of writing $(\xi_1, \xi_2) \in \Rightarrow_{G_0}$, we write $\xi_1 \Rightarrow_{G_0} \xi_2$. □

Definition 9 (Language Generated by Context-free Grammar). Let $G_0 = (N, \Lambda, Z, P)$ be a context-free grammar. The *language generated by* G_0, denoted by $L(G_0)$, is the set $\{\lambda \in \Lambda^* \mid Z \Rightarrow^*_{G_0} \lambda\}$. □

3.2 Sorted Regular Tree Grammars

Definition 10 (S-sorted Regular Tree Grammar). An *S-sorted regular tree grammar* is a tuple $G = (S, N, \Sigma, Z, R)$, where:

- S is a (non-empty) finite set of *sorts*,

- N is an S-sorted set of *nonterminal symbols*,
- Σ is an S-sorted ranked alphabet of *terminal symbols*, where $\Sigma \cap N = \emptyset$,
- $Z \in N$ is the *start symbol*,
- R is a finite set of *rules* of the form $A \rightarrow t$ with $A \in N^s$ and $te \in T_\Sigma(N)^s$ for a sort $s \in S$. $\square$

Definition 11 (Derivation Relation of S-sorted Regular Tree Grammar). Let $G = (S, N, \Sigma, Z, R)$ be an S-sorted regular tree grammar. The *derivation relation of* G, denoted by $\Rightarrow_G$, is the smallest binary relation $\Rightarrow \subseteq T_\Sigma(N) \times T_\Sigma(N)$ such that for every $te_1, te_2 \in T_\Sigma(N)$, $(te_1, te_2) \in \Rightarrow$ if there exist a rule $A \rightarrow te \in R$ and an occurrence of $A \in N$ in te_1 such that te_2 is obtained from te_1 by replacing this occurrence of A in te_1 with $te \in T_\Sigma(N)$. Instead of writing $(te_1, te_2) \in \Rightarrow_G$, we write $te_1 \Rightarrow_G te_2$. $\square$

Definition 12 (Language Generated by S-sorted Regular Tree Grammar). Let $G = (S, N, \Sigma, Z, R)$ be an S-sorted regular tree grammar. The *language generated by* G, denoted by $L(G)$, is the set $\{te \in T_\Sigma \mid Z \Rightarrow_G^* te\}$. $\square$

3.3 Relatedness of Context-free and Regular Tree Grammars

Definition 13 (Context-free and Regular Tree Grammar Relatedness). Let $G_0 = (N, \Lambda, Z, P)$ be a context-free grammar and $\pi : P \rightarrow \Sigma$ be a bijection, where Σ is a sorted ranked alphabet of *production names* whose cardinality $card(\Sigma) = card(P)$. The regular tree grammar which is *related to* G_0 *by* π is the *N-sorted regular tree grammar* $G = (N, N, \Sigma, Z, R_P)$ containing the finite set N of sorts, N-sorted set N of non-terminal symbols, N-sorted ranked alphabet Σ of terminal symbols (production names), the start symbol Z, and the smallest set R_P of rules, such that the following holds:

- for every non-terminal symbol $\eta \in N$, $sort(\eta) = \eta$,
- for every production $\eta_0 \rightarrow \lambda_0\eta_1\lambda_1 \dots \eta_k\lambda_k \in P$ with $k \geq 0$, $\eta_0, \dots, \eta_k \in N$, and $\lambda_0, \dots, \lambda_k \in \Lambda^*$ and $\pi(\eta_0 \rightarrow \lambda_0\eta_1\lambda_1 \dots \eta_k\lambda_k) = \sigma$ with $\sigma \in \Sigma$, we have $sort(\sigma) = (\eta_1 \dots \eta_k, \eta_0)$ and $\eta_0 \rightarrow \sigma(\eta_1, \dots, \eta_k) \in R_P$.

$\square$

3.4 Abstract Syntax Trees

Definition 14 (Abstract Syntax Trees). Let $G_0 = (N, \Lambda, Z, P)$ be a context-free grammar and $G = (N, N, \Sigma, Z, R_P)$ be the regular tree grammar which is related to G_0 by a bijection $\pi : P \rightarrow \Sigma$. The set of *abstract syntax trees* of G_0 regarding π, denoted by $AST(G_0, \pi)$, is the set of all N-sorted trees with the sort Z over the N-sorted ranked alphabet Σ, i.e. $AST(G_0, \pi) = TR(\Sigma)^Z$. $\square$

Note that for an N-sorted regular tree grammar $G = (N, N, \Sigma, Z, R_P)$ which is related to a context-free grammar $G_0 = (N, \Lambda, Z, P)$ by a bijection $\pi : P \rightarrow \Sigma$, $TR(\Sigma)^Z = \{t \mid t \in gr(te) \text{ where } te \in L(G)\}$, since the productions P in G_0 are mapped bijectively to the N-sorted ranked alphabet Σ in G.

Chapter 4

Interface to The Attribute Grammar

In chapter 6 we shall discuss a variant of the monadic second order (MSO) logic, namely *MSO** logic. This variant combines the formalisms of MSO logic and the (macro) attribute grammar [4, 6]. In order to have an interface from the MSO logic formalism into the world of attribute grammar, in this chapter we introduce some necessary notions and notations.

Let K be a non-empty finite set of sorts, Ω be a K-sorted set of values, $G_0 = (N, \Lambda, Z, P)$be a context-free grammar and $G = (N, N, \Sigma, Z, R_P)$ be the N-sorted regular tree grammar, which is related to G_0 by a bijection $\pi : P \to \Sigma$.

By an attribute grammar, the nodes of an abstract syntax tree of the context-free grammar G_0 regarding π are also equipped with attributes. Those attributes, together with functions over a semantic domain, are defined in the attribute grammar. By evaluating the attribute values, one can express a property between nodes whose attribute values are evaluated. This is useful in particular when we want to specify context-sensitive phenomena of a formal (programming) language.

In the following we define an interface that enables us to relate the formalisms of MSO logic and the attribute grammar.

Definition 15 (Attribute Symbols). Let $\mathcal{A}$ be a $(\mathcal{P}(N) \times K^* \times K)$-sorted set (of *attribute symbols*). For every $a \in \mathcal{A}^{(\widetilde{\eta}, \kappa_1 \dots \kappa_k, \kappa_0)}$ with $\widetilde{\eta} \in \mathcal{P}(N)$, $k \geq 0$, and $\kappa_0, \dots, \kappa_k \in K$, we define the functions $sort_{nodes} : \mathcal{A} \to \mathcal{P}(N)$ and $sort_{fun} : \mathcal{A} \to (K^* \times K)$, such that $sort_{nodes}(a) = \widetilde{\eta}$ and $sort_{fun}(a) = (\kappa_1 \dots \kappa_k, \kappa_0)$.

Let $\mathcal{V}_1$ be a $\mathcal{P}(N)$-sorted set (of node variables). For $\mathcal{V}_1$, we define the K-sorted ranked alphabet (of *attribute variable symbols*) as follows:

$$\mathcal{I}\langle \mathcal{A}, \mathcal{V}_1 \rangle = \{ \langle a, x \rangle^{sort_{fun}(a)} \mid a \in \mathcal{A},\ x \in \mathcal{V}_1,\ sort(x) \subseteq sort_{nodes}(a) \}.$$

Let V be a finite N-sorted set (of nodes). For V, we define the K-sorted ranked alphabet (of *attribute instance symbols*) as follows:

$$\mathcal{I}\langle\mathcal{A}, V\rangle = \{\langle a, n\rangle^{sort_{fun}(a)} \mid a \in \mathcal{A},\ n \in V,\ sort(n) \in sort_{nodes}(a)\}.$$

$\square$

Definition 16 (Semantics of Attribute Instance Symbols). Let $t \in TR(\Sigma)$ be an (abstract syntax) tree. For every $\langle a, n\rangle \in \mathcal{I}\langle\mathcal{A}, V_t\rangle^{(\kappa_1 \dots \kappa_k, \kappa_0)}$ with $k \geq 0$ and $\kappa_0, \dots, \kappa_k \in K$:

$$dec_t(\langle a, n\rangle) : \Omega^{\kappa_1} \times \ldots \times \Omega^{\kappa_k} \rightarrow \Omega^{\kappa_0}$$

is a k-ary computable function. $\square$

Definition 17 (Function Symbols and Their Semantics). Let $\mathcal{F}$ be a K-sorted ranked alphabet (of *function symbols*). For every $r \geq 0$, $\kappa_0, \dots, \kappa_r$, and $f \in \mathcal{F}^{(\kappa_1 \dots \kappa_r, \kappa_0)}$:

$$int_{\mathcal{F}}(f) : \Omega^{\kappa_1} \times \ldots \times \Omega^{\kappa_r} \rightarrow \Omega^{\kappa_0}$$

is an r-ary computable function. $\square$

We also give the following definition which is needed later when we discuss the semantics of MSO* logic.

Definition 18 (Interpretation of Terms over $\langle\mathcal{A}, \mathcal{V}_1\rangle$ and $\mathcal{F}$). For a given (abstract syntax) tree $t \in TR(\Sigma)$, an interpretation function dec_t, an interpretation function $int_{\mathcal{F}}$ for function symbols, and a function $\theta : \mathcal{V}_1 \rightarrow V_t$ that assigns every node variable $x \in \mathcal{V}_1$ a node $n \in V_t$ (where $sort(n) \in sort(x)$), we define the function $\tau_\theta(dec_t, int_{\mathcal{F}}) : T_{\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}} \rightarrow \Omega$ that assigns every term $te \in T_{\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}$ a semantic value $\omega \in \Omega$, where te and ω have the same sort. It is defined as the following function τ by induction on the structure of $T_{\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}$:

- $te = f(te_1, \dots, te_r)$,
  where $r \geq 0$, $\kappa_0, \dots, \kappa_r \in K$, $f \in \mathcal{F}^{(\kappa_1 \dots \kappa_r, \kappa_0)}$, and
  $te_1 \in T^{\kappa_1}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}, \dots, te_r \in T^{\kappa_r}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}$:
  $\tau(te) = int_{\mathcal{F}}(f)(\tau(te_1), \dots, \tau(te_r))$

- $te = \langle a, x\rangle(te_1, \dots, te_k)$,
  where $k \geq 0$, $\kappa_0, \dots, \kappa_k \in K$, $\widetilde{\eta} \in \mathcal{P}(N)$, $a \in \mathcal{A}^{(\widetilde{\eta}, \kappa_1 \dots \kappa_k, \kappa_0)}$, $x \in \mathcal{V}_1$, $sort(x) \subseteq \widetilde{\eta}$, and $te_1 \in T^{\kappa_1}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}, \dots, te_k \in T^{\kappa_k}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}$:
  $\tau(te) = dec_t(\langle a, \theta(x)\rangle)(\tau(te_1), \dots, \tau(te_k))$

$\square$

Chapter 5

Bottom-up Tree Automata

In this chapter we define the bottom-up tree automata, which will be used to give the operational semantics of MSO^* logic.[1] In our definition of bottom-up tree automata, we consider infinite set of states which is needed when we deal with the attribute formula. We also define the final states slightly different from what usually found in the literature. We define the final states, not only as a set of states, but as a family of sets of states.

In the following, the sorted set N and the N-sorted ranked alphabet Σ come from the N-sorted regular tree grammar (which is related to a context-free grammar by a bijection mapping from its productions to sorted names) from which our abstract syntax trees are built. Let K be a non-empty finite set of sorts. Let also $\mathcal{A}$ be a $(\mathcal{P}(N) \times K^* \times K)$-sorted set (of attribute symbols) and $\mathcal{F}$ be a K-sorted ranked alphabet (of function symbols). Given the interpretation function $int_{\mathcal{F}}$ and a family $dec = \{dec_t | t \in TR(\Sigma)\}$ of interpretation functions, we define in the following both the deterministic and the nondeterministic bottom-up tree automata.

5.1 Deterministic Bottom-Up Tree Automata

We define what is meant by a deterministic bottom-up tree automaton and the language accepted by it.

Definition 19 (Deterministic Bottom-Up Tree Automata). A *deterministic bottom-up tree automaton* $\mathcal{M}_{dec,int_{\mathcal{F}}}$ is a tuple:

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma, \delta, F),$$

where:

- Q is an infinite *set of states*,
- Σ is an N*-sorted ranked alphabet*,
- $\delta = \{\delta^k_\sigma\}_{\sigma \in \Sigma^{(\eta_1 \cdots \eta_k, \eta_0)}, k \geq 0}$ is a family of *transition functions*, where:

$$\delta^k_\sigma : Q^k \to Q,$$

[1]For short, in this report we also write *automata* to mean *bottom up tree automata.*

- $F = \{F_{dec_t,int_{\mathcal{F}}}\}_{t \in TR(\Sigma)}$ is a family of *sets of final states*, where for every $t \in TR(\Sigma)$, $F_{dec_t,int_{\mathcal{F}}} \subseteq Q$.

We also define $\widehat{\delta} : TR(\Sigma) \to Q$ to compute the run of a deterministic bottom-up tree automata on a tree $t \in TR(\Sigma)$ by structural induction on $TR(\Sigma)$ as follows:

- <u>if</u> $t = (V_t, E_t, lab_t) \in TR(\Sigma)$, where:
  $V_t = \{n^{\eta_0}\}$ with $\eta_0 \in N$, $E_t = \emptyset$, and $lab_t(n) \in \Sigma^{(\varepsilon,\eta_0)}$
  <u>then</u>:
  $$\widehat{\delta}(t) = \delta^0_{lab_t(n)}\ (\ )$$

- <u>if</u> $t = (V_t, E_t, lab_t) \in TR(\Sigma)$ and there are $t_i = (V_{t_i}, E_{t_i}, lab_{t_i}) \in TR(\Sigma)$, where $i \in [k]$, $k > 0$, such that:

  - $V_t = \{n^{\eta_0}\} \cup \bigcup_{i \in [k]} V_{t_i}$ with $\eta_0 \in N$,
  - $E_t = \{(n, i, root(t_i)) \mid i \in [k]\} \cup \bigcup_{i \in [k]} E_{t_i}$,
  - $lab_t(n) \in \Sigma^{(\eta_1 \cdots \eta_k, \eta_0)}$ with $\eta_0, \ldots, \eta_k \in N$ and for every $m \in V_{t_i}$ with $i \in [k]$, $lab_t(m) = lab_{t_i}(m)$,

  <u>then</u>:
  $$\widehat{\delta}(t) = \delta^k_{lab_t(n)}\ (\widehat{\delta}(t_1), \ldots, \widehat{\delta}(t_k))$$

$\square$

Definition 20 (Language Accepted by Deterministic Bottom-Up Tree Automata). Let $\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma, \delta, F)$ be a deterministic bottom-up tree automaton. The *language accepted by* $\mathcal{M}_{dec,int_{\mathcal{F}}}$, denoted by $L(\mathcal{M}_{dec,int_{\mathcal{F}}})$, is defined as follows:

$$L(\mathcal{M}_{dec,int_{\mathcal{F}}}) = \{t \in TR(\Sigma) \mid \widehat{\delta}(t) \in F_{dec_t,int_{\mathcal{F}}}\}.$$

$\square$

5.2 Nondeterministic Bottom-Up Tree Automata

We define what is meant by a nondeterministic bottom-up tree automaton and the language accepted by it.

Definition 21 (Nondeterministic Bottom-Up Tree Automata). A *nondeterministic bottom-up tree automaton* $\mathcal{M}_{dec,int_{\mathcal{F}}}$ is a tuple:

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma, \delta, F),$$

where:

- Q is an infinite *set of states*,
- Σ is an N*-sorted ranked alphabet*,
- $\delta = \{\delta^k_\sigma\}_{\sigma \in \Sigma^{(\eta_1 \cdots \eta_k, \eta_0)}, k \geq 0}$ is a family of *transition functions*, where:

$$\delta^k_\sigma : Q^k \to \mathcal{P}(Q),$$

- $F = \{F_{dec_t,int_{\mathcal{F}}}\}_{t \in TR(\Sigma)}$ is a family of *sets of final states*, where for every $t \in TR(\Sigma)$, $F_{dec_t,int_{\mathcal{F}}} \subseteq Q$.

We also define $\widehat{\delta} : TR(\Sigma) \rightarrow \mathcal{P}(Q)$ to compute the run of a nondeterministic bottom-up tree automata on a tree $t \in TR(\Sigma)$ by structural induction on $TR(\Sigma)$ as follows:

- <u>if</u> $t = (V_t, E_t, lab_t) \in TR(\Sigma)$, where:
  $V_t = \{n^{\eta_0}\}$ with $\eta_0 \in N$, $E_t = \emptyset$, and $lab_t(n) \in \Sigma^{(\varepsilon,\eta_0)}$
  <u>then</u>:
  $$\widehat{\delta}(t) = \delta^0_{lab_t(n)}\ ()$$
- <u>if</u> $t = (V_t, E_t, lab_t) \in TR(\Sigma)$ and there are $t_i = (V_{t_i}, E_{t_i}, lab_{t_i}) \in TR(\Sigma)$, where $i \in [k]$, $k > 0$, such that:
  - $V_t = \{n^{\eta_0}\} \cup \bigcup_{i\in[k]} V_{t_i}$ with $\eta_0 \in N$,
  - $E_t = \{(n, i, root(t_i)) \mid i \in [k]\} \cup \bigcup_{i\in[k]} E_{t_i}$,
  - $lab_t(n) \in \Sigma^{(\eta_1\cdots\eta_k,\eta_0)}$ with $\eta_0, \ldots, \eta_k \in N$ and for every $m \in V_{t_i}$ with $i \in [k]$, $lab_t(m) = lab_{t_i}(m)$,

  <u>then</u>:
  $$\widehat{\delta}(t) = \bigcup_{(q_1,\ldots,q_k)\in\widehat{\delta}(t_1)\times\ldots\times\widehat{\delta}(t_k)} \delta^k_{lab_t(n)}\ (q_1, \ldots, q_k)$$

□

Definition 22 (Language Accepted by Nondeterministic Bottom-Up Tree Automata). Let $\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma, \delta, F)$ be a nondeterministic bottom-up tree automaton. The *language accepted by* $\mathcal{M}_{dec,int_{\mathcal{F}}}$, which is denoted by $L(\mathcal{M}_{dec,int_{\mathcal{F}}})$, is defined as follows:

$$L(\mathcal{M}_{dec,int_{\mathcal{F}}}) = \{t \in TR(\Sigma) \mid \widehat{\delta}(t) \cap F_{dec_t,int_{\mathcal{F}}} \neq \emptyset\}.$$

□

5.3 Relationship between Deterministic and Nondeterministic Bottom-Up Tree Automata

We recall here how a nondeterministic bottom-up finite state tree automaton is related to a deterministic bottom-up finite state tree automaton, where a finite state tree automaton is an automaton which has a finite number of states. The relationship between them is established by the so-called *power set construction.* This construction is used to construct a deterministic bottom up finite state tree automaton from a nondeterministic bottom up finite state tree automaton. It has been proved that by this construction the language which is accepted by the nondeterministic bottom up finite state tree automaton is also accepted by the deterministic bottom up finite state tree automata.

Let $\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma, \delta, F)$ be a *nondeterministic* bottom up finite state tree automaton. We construct the *deterministic* bottom up finite state tree automaton $\mathcal{M}'_{dec,int_{\mathcal{F}}} = (Q', \Sigma, \delta', F')$, where Q', δ' and F' are defined as follows:

- $Q' = \mathcal{P}(Q)$,
- $\delta' = \{\delta'^{k}_{\sigma}\}_{\sigma \in \Sigma^{(\eta_1 \cdots \eta_k, \eta_0)}, k \geq 0}$, where $\delta'^{k}_{\sigma} : \mathcal{P}(Q)^k \to \mathcal{P}(Q)$ is defined as follows:

  For every $P_1, \ldots, P_k \in Q'$:

  $$\delta'^{k}_{\sigma}(P_1, \ldots, P_k) = \bigcup_{(q_1,\ldots,q_k) \in P_1 \times \ldots \times P_k} \delta^{k}_{\sigma}(q_1, \ldots, q_k),$$

- $F' = \{F'_{dec_t, int_{\mathcal{F}}}\}_{t \in TR(\Sigma)}$, where for every $t \in TR(\Sigma)$:
  $F'_{dec_t, int_{\mathcal{F}}} = \{P \in Q' \mid P \cap F_{dec_t, int_{\mathcal{F}}} \neq \emptyset\}$.

And we have, that $L(\mathcal{M}_{dec, int_{\mathcal{F}}}) = L(\mathcal{M}'_{dec, int_{\mathcal{F}}})$.

We shall later use this construction in our work. Although we deal with infinite number of states of bottom up tree automata, we can still use the idea of this construction to construct a deterministic bottom up tree automaton from a nondeterministic one. Instead of applying the construction naively (which will not work, since the number of states is infinite), in section 7.2 we propose a special representation of states for an automaton with infinite number of states, where we represent the states symbolically. Moreover, in the construction we shall consider only the reachable states, instead of all states.

Chapter 6

MSO* Logic

In this chapter we discuss a variant of the monadic second order (MSO) logic, namely *MSO** logic. This variant combines the attribute grammar and the MSO logic formalism.

We give the syntax of the MSO* logic by introducing appropriate formulas to define relations over nodes or set of nodes of an abstract syntax tree. In particular, we define a term as an atomic formula, where this term is built from attribute variable symbols and function symbols and this term can be interpreted as a boolean value. We name this special formula an *attribute formula*. With this formula we are able to relate the attribute grammar and the MSO logic formalism. Hence, by introducing this formula into MSO logic, one is not only able to specify context-free phenomena, but also context-sensitive phenomena. In spite of this advantage, adding this formula into the logic raises some issues and problems, in particular when we deal with the operational semantics of the logic, where we need to use a bottom-up tree automaton with *infinite* number of states. Therefore, a special handling is needed for the representation and the inductive construction of the automata. We shall discuss this in more detail in the subsequent chapter.

To define an attribute formula, one needs an interface between the attribute grammar and the MSO logic formalism. Such interface has been introduced in chapter 4. We need the sorted set $\mathcal{A}$ and the sorted ranked alphabet $\mathcal{F}$ as an interface to the attributes and functions over a semantic domain, respectively, which are defined in an attribute grammar. We use a family *dec* of interpretation functions dec_t for every abstract syntax tree t as an interface to evaluate the attribute values of nodes in t (also known as *decoration* of t in the attribute grammar formalism). And every function f over a semantic domain which is used in an attribute formula can be computed by means of $int_{\mathcal{F}}$ as the interface.

We start our discussion in this chapter by giving the syntax of MSO* logic and then its semantics, both the model theoretical semantics and the operational semantics. As we have mentioned, we shall use the bottom-up tree automata (with infinite number of states) to define the operational semantics of the MSO* logic.

In the following, the sorted set N and the N-sorted ranked alphabet Σ come from the N-sorted regular tree grammar (which is related to a context-free grammar by a bijection mapping from its productions to sorted names) from which our abstract syntax trees are built. Let $\mathcal{V}_1$ be a $\mathcal{P}(N)$-sorted set of first order variables (or *node variables*), and $\mathcal{V}_2$ be a $\mathcal{P}(N)$-sorted set of second order variable (or *node set variables*).[1]

6.1 Syntax

Let K be a non-empty finite set of sorts. Let also $\mathcal{A}$ be a $(\mathcal{P}(N) \times K^* \times K)$-sorted set of attribute symbols and $\mathcal{F}$ be a K-sorted ranked alphabet of function symbols.

Definition 23 (Set of MSO* Formulas). We denote the *set of MSO* formulas* by $MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, where for every formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, the set of free first order variables in φ is a subset of $\mathcal{V}_1$ and the set of free second order variables in φ is a subset of $\mathcal{V}_2$. Additionally, no bound first order variable (and second order variables, respectively) of φ is in $\mathcal{V}_1$ (and $\mathcal{V}_2$, respectively).

We define $MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ as the smallest set consisting the following formulas:

1. the *atomic formulas*:
   - $true$,
   - $label_\sigma(x)$, where $x \in \mathcal{V}_1$, $\sigma \in \Sigma^{(\eta_1\cdots\eta_k,\eta_0)}, k \geq 0, \eta_0, \ldots, \eta_k \in N$, and $\eta_0 \in sort(x)$,
   - $x \in X$, where $x \in \mathcal{V}_1$, $X \in \mathcal{V}_2$, and $sort(x) = sort(X)$,
   - $x == y$, where $x, y \in \mathcal{V}_1$ and $sort(x) = sort(y)$,
   - $type_\eta(x)$, where $x \in \mathcal{V}_1$ and $\eta \in sort(x)$,
   - $edge_i(x, y)$, where $x, y \in \mathcal{V}_1$ and $1 \leq i \leq max\_rank(\Sigma)$,
   - $x\ over\ y$, where $x, y \in \mathcal{V}_1$,
   - $\varphi \in T^{Bool}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}$ (*attribute formula*)
2. the *built formulas*:
   - $\neg\psi$, where $\psi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$,
   - $\psi_1 \wedge \psi_2$, where $\psi_1, \psi_2 \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$,
   - $\exists x : \widetilde{\eta}.\psi$, where $x \notin \mathcal{V}_1$, $\widetilde{\eta} \in \mathcal{P}(N)$, and $\psi \in MSO^*(\Sigma, \mathcal{V}_1 \cup \{x^{\widetilde{\eta}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$,
   - $\exists X : \widetilde{\eta}.\psi$, where $X \notin \mathcal{V}_2$, $\widetilde{\eta} \in \mathcal{P}(N)$, and $\psi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2 \cup \{X^{\widetilde{\eta}}\})_{\mathcal{A},\mathcal{F}}$.

   In the following we define the formulas $false$, $x\ under\ y$, the logical connectives $\vee$, $\rightarrow$, $\leftrightarrow$, and the universal quantifier $\forall$ by using the atomic formulas $true$, $x\ over\ y$, the logical connectives $\neg$, $\wedge$, and the existential quantifier $\exists$:

   - $false = \neg true$,
   - $(x\ under\ y) = (y\ over\ x)$,

[1] We use small letters, e.g. $x, y, \ldots$ to denote first order variables and capital letters, e.g. $X, Y, \ldots$ to denote second order variables.

- $(\psi_1 \vee \psi_2) = \neg(\neg\psi_1 \wedge \neg\psi_2)$,
- $(\psi_1 \rightarrow \psi_2) = \neg(\psi_1 \wedge \neg\psi_2)$,
- $(\psi_1 \leftrightarrow \psi_2) = (\psi_1 \rightarrow \psi_2) \wedge (\psi_2 \rightarrow \psi_1)$,
- $(\forall x : \widetilde{\eta}.\psi) = \neg(\exists x : \widetilde{\eta}.\neg\psi)$,
- $(\forall X : \widetilde{\eta}.\psi) = \neg(\exists X : \widetilde{\eta}.\neg\psi)$.

$\square$

We choose to have those atomic formulas with the following motivation:

- The formula *true* (as a boolean constant) is useful for example, if we want to know whether there is a node in an abstract syntax tree having a particular sort η. This can be formulated as $\exists x : \{\eta\}.\ true$.
- The formulas $label_\sigma(x)$ and $type_\eta(x)$ provide the ability to test a node whether it qualifies a property, i.e. the label of a node (for the first formula) and the sort of a node (for the latter formula). The formula $x \in X$ can also be seen as a test for a node x, whether it qualifies the property described by X.
- The formulas $x == y$, $edge_i(x, y)$, and $x\ over\ y$ provide the ability to express positional property between two nodes in an abstract syntax tree, i.e. whether two nodes are in the same position (for the formula $x == y$), whether a node is the i-th (direct) successor of another node (for the formula $edge_i(x, y)$), and whether a node is an indirect successor of another node (for the formula $x\ over\ y$). The formula $x == y$ can be considered as a macro for the formula $\forall X : \widetilde{\eta}.\ ((x \in X) \leftrightarrow (y \in X))$. But, having $x == y$ as an atomic formula turns out to be more efficient in defining its decision procedure based on the operational semantics which will be discussed later.
- The attribute formula provides the ability to test the relationship between nodes which is established by the evaluation of the attribute values of the nodes. By introducing this formula, we are not only able to test a simple property of nodes, e.g. the sort of nodes or the label of nodes, but we are also able to test a more subtle property of nodes which are expressed by means of their attributes.

Note that for built formulas using quantifiers, our syntactic restrictions consequently disallow the following:

- variables to occur both free and bound at the same time. For example, consider the formula:

  $$\exists y : \widetilde{\eta_1}.((\exists x : \widetilde{\eta_2}.label_\sigma(x)) \wedge edge_1(x, y)).$$

  According to our syntactic restriction of quantified formula, we have:

  $$((\exists x : \widetilde{\eta_2}.label_\sigma(x)) \wedge edge_1(x, y)) \in MSO^*(\Sigma, \mathcal{V}_1 \cup \{y^{\widetilde{\eta_1}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

  Let $\psi_1 = \exists x : \widetilde{\eta_2}.label_\sigma(x)$ and $\psi_2 = edge_1(x, y)$. Now, by the syntactic restriction of conjunction, $\psi_1, \psi_2 \in MSO^*(\Sigma, \mathcal{V}_1 \cup \{y^{\widetilde{\eta_1}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. If we consider the subformula ψ_1, then $x^{\widetilde{\eta_2}} \notin \mathcal{V}_1$. But, on the other hand, if we consider the subformula ψ_2, then $x^{\widetilde{\eta_2}} \in \mathcal{V}_1$. Thus, we have a contradiction in this case.

- in case of nested quantified formulas, variables to be bound more than once at the same time. For example, consider the formula:

  $$\exists x : \widetilde{\eta}.(label_\alpha(x) \wedge \exists x : \widetilde{\eta}.label_\beta(x)).$$

  According to our syntactic restriction of quantified formula, we have:

  $$(label_\alpha(x) \wedge \exists x : \widetilde{\eta}.label_\beta(x)) \in MSO^*(\Sigma, \mathcal{V}_1 \cup \{x^{\widetilde{\eta}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

  Let $\psi_1 = label_\alpha(x)$ and $\psi_2 = \exists x : \widetilde{\eta}.label_\beta(x)$. Now, by the syntactic restriction of conjunction, $\psi_1, \psi_2 \in MSO^*(\Sigma, \mathcal{V}_1 \cup \{x^{\widetilde{\eta}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. But, on the other hand, because of the syntactic restriction of quantified formula, we also have $\psi_2 \notin MSO^*(\Sigma, \mathcal{V}_1 \cup \{x^{\widetilde{\eta}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Thus, we have a contradiction in this case.

Note that in case of non-nested quantified formulas, variables are allowed to be bound more than once at the same time. Consider the following example:

$$(\exists x : \widetilde{\eta}.label_\alpha(x)) \wedge (\exists x : \widetilde{\eta}.label_\beta(x)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2),$$

where $x^{\widetilde{\eta}} \notin \mathcal{V}_1$. Let $\psi_1 = \exists x : \widetilde{\eta}.label_\alpha(x)$ and $\psi_2 = \exists x : \widetilde{\eta}.label_\beta(x)$. According to the syntactic restriction of conjunction, we have $\psi_1, \psi_2 \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, where $x^{\widetilde{\eta}} \notin \mathcal{V}_1$. And this is also consistent with the syntactic restriction of a quantified formula.

6.2 Semantics

Before we define the semantics of MSO* logic, we give the definitions of the extended alphabets and well-marked trees.

Definition 24 (Extended Alphabet). The *alphabet* Σ *extended by* $\mathcal{V}_1$ *and* $\mathcal{V}_2$, denoted by $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$, is the set:

$$\begin{aligned}\Sigma_{\mathcal{V}_1,\mathcal{V}_2} \quad = \quad & \{(\sigma, U_1, U_2)^{(\eta_1 \cdots \eta_k, \eta_0)} | \sigma \in \Sigma^{(\eta_1 \cdots \eta_k, \eta_0)}, k \geq 0, U_1 \subseteq \bigcup_{\eta_0 \in \widetilde{\eta}} \mathcal{V}_1^{\widetilde{\eta}}, \\ & U_2 \subseteq \bigcup_{\eta_0 \in \widetilde{\eta}} \mathcal{V}_2^{\widetilde{\eta}}\}\end{aligned}$$

The elements of $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$ are called *input symbols.*

The node-labeling function defined in definition 3 is also extended. For every $t \in TR(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})$ and $n \in V_t$ with $lab_t(n) = (\sigma, U_1, U_2)$, the functions $lab_t^\Sigma : V_t \to \Sigma$, $lab_t^{1^{st}} : V_t \to \mathcal{P}(\mathcal{V}_1)$, and $lab_t^{2^{nd}} : V_t \to \mathcal{P}(\mathcal{V}_2)$ are defined as follows:

- $lab_t^\Sigma(n) = \sigma$
- $lab_t^{1^{st}}(n) = U_1$
- $lab_t^{2^{nd}}(n) = U_2$

We also define for every $t = (V_t, E_t, lab_t) \in TR(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})$ the function $peel_{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)} : TR(\Sigma_{\mathcal{V}_1,\mathcal{V}_2}) \to TR(\Sigma)$, such that $peel_{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}(t) = (V_t, E_t, lab'_t)$, where for every $n \in V_t$, $lab'_t(n) = lab_t^\Sigma(n)$. We leave out the indices Σ, $\mathcal{V}_1$, and $\mathcal{V}_2$, whenever they are clear from the context. □

Definition 25 (Well-marked Trees). Let $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$ be an extended alphabet. The *set of well-marked trees over* $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$, denoted by $WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, is the set:

$$WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2) = \{t \in TR(\Sigma_{\mathcal{V}_1,\mathcal{V}_2}) | \forall x \in \mathcal{V}_1 . \exists! n \in V_t : x \in lab_t^{1^{st}}(n)\},$$

where the quantifier $\exists!$ is to be read *"there exists exactly one"*. $\square$

Given the interpretation function $int_{\mathcal{F}}$ and a family $dec = \{dec_t | t \in TR(\Sigma)\}$ of interpretation functions, in the following we define the *model theoretical* semantics and the *operational* semantics of MSO* logic.

6.2.1 Model Theoretical Semantics

We define the model theoretical semantics of MSO* logic via model operator and then we define the tree language defined by an MSO* formula.

Definition 26 (Model Operator). We define the *model operator*:

$$\models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \subseteq WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$$

by induction on the structure of the MSO* formulas. In the following, we write the model operator $\models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}}$ as an infix operator.

Let $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$. We define $\models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}}$ as follows:

- $\varphi = true \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \varphi$,
- $\varphi = (label_\sigma(x)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \varphi$ iff n is the node of t such that $x \in lab_t^{1^{st}}(n)$ and $lab_t^{\Sigma}(n) = \sigma$,
- $\varphi = (x \in X) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \varphi$ iff n is the node of t such that $x \in lab_t^{1^{st}}(n)$ and $X \in lab_t^{2^{nd}}(n)$,
- $\varphi = (x == y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \varphi$ iff n is the node of t such that $x \in lab_t^{1^{st}}(n)$ and $y \in lab_t^{1^{st}}(n)$,
- $\varphi = (type_\eta(x)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \varphi$ iff n is the node of t such that $x \in lab_t^{1^{st}}(n)$ and $sort(n) = \eta$,
- $\varphi = (edge_i(x, y)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \varphi$ iff n_1 and n_2 are the nodes of t such that $x \in lab_t^{1^{st}}(n_1)$, $y \in lab_t^{1^{st}}(n_2)$, and $(n_1, i, n_2) \in E_t$,
- $\varphi = (x\ over\ y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_{\mathcal{F}}} \varphi$ iff n_1 and n_2 are the nodes of t such that $x \in lab_t^{1^{st}}(n_1)$, $y \in lab_t^{1^{st}}(n_2)$, and there is a sequence $a_1, \ldots, a_n$ of nodes with $n > 1$ such that $(a_i, j_i, a_{i+1}) \in E_t$ for every $1 \leq i < n$ and some $j_i \in [rank(lab_t(a_i))]$ with $a_1 = n_1$ and $a_n = n_2$,

- $(\varphi \in T^{Bool}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \varphi$ iff for every $x \in \mathcal{V}_1$, $\theta(x)$ is the node of t such that $x \in lab^{1^{st}}_t(\theta(x))$ and $\tau_\theta(dec_{peel(t)}, int_\mathcal{F})(\varphi) = True$,

- $\varphi = (\neg\psi) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \varphi$ iff $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \psi$ is not true,

- $\varphi = (\psi_1 \wedge \psi_2) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \varphi$ iff $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \psi_1$ and $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \psi_2$,

- $\varphi = (\exists x : \widetilde{\eta}.\psi) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  Note that $x^{\widetilde{\eta}} \notin \mathcal{V}_1$ and $\psi \in MSO^*(\Sigma,\mathcal{V}_1 \cup \{x^{\widetilde{\eta}}\},\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$.
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \varphi$ iff there is a node $n \in \bigcup_{\eta\in\widetilde{\eta}} V^\eta_t$ such that $t' \models^{(\Sigma,\mathcal{V}_1\cup\{x^{\widetilde{\eta}}\},\mathcal{V}_2)}_{dec,int_\mathcal{F}} \psi$, where $t' \in WTR(\Sigma,\mathcal{V}_1 \cup \{x^{\widetilde{\eta}}\},\mathcal{V}_2)$ is obtained from t by defining $lab^{1^{st}}_{t'}(n) = lab^{1^{st}}_t(n) \cup \{x^{\widetilde{\eta}}\}$ and keeping the labels of other nodes in t unchanged,

- $\varphi = (\exists X : \widetilde{\eta}.\psi) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$:
  Note that $X^{\widetilde{\eta}} \notin \mathcal{V}_2$ and $\psi \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2 \cup \{X^{\widetilde{\eta}}\})_{\mathcal{A},\mathcal{F}}$.
  $t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \varphi$ iff there is a set of nodes $\{n_1,\ldots,n_k\} \subseteq \bigcup_{\eta\in\widetilde{\eta}} V^\eta_t$ such that $t' \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2\cup\{X^{\widetilde{\eta}}\})}_{dec,int_\mathcal{F}} \psi$, where $t' \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2 \cup \{X^{\widetilde{\eta}}\})$ is obtained from t by defining $lab^{2^{nd}}_{t'}(n_i) = lab^{2^{nd}}_t(n_i) \cup \{X^{\widetilde{\eta}}\}$ for every $1 \le i \le k$, and keeping the labels of other nodes in t unchanged.

□

Definition 27 (Tree language defined by an MSO* formula). Let $\varphi \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Given the interpretation function $int_\mathcal{F}$ and a family $dec = \{dec_t | t \in TR(\Sigma)\}$ of interpretation functions, the *tree language defined by* φ, denoted by $L^{dec,int_\mathcal{F}}_\varphi$, is the set:

$$L^{dec,int_\mathcal{F}}_\varphi = \{t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2) | t \models^{(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}_{dec,int_\mathcal{F}} \varphi\}$$

□

6.2.2 Operational Semantics

We shall give the operational semantics of MSO* logic using the bottom-up tree automaton which was defined in chapter 5 by giving the construction of the automata for the atomic and built formulas of MSO* logic. In the construction of the automata, we assume that the given trees on which the automaton runs are *well-marked trees*. This assumption holds for the automaton of every MSO* formula, i.e. atomic formulas and built formulas. This assumption can be made by considering the following justification. For atomic formulas, we just let the automata run on well-marked trees. In case of built formulas:

- given a formula $\varphi = (\neg\psi) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, we let the automaton of ψ run on well-marked trees,

- given a formula $\varphi = (\psi_1 \wedge \psi_2) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, we let the automata for ψ_1 and ψ_2 run in parallel on well-marked trees,

- given a formula with quantifier, in the run of their automata we guess the value of the quantified variable on well-marked trees:
  - for a formula $\varphi = (\exists x : \widetilde{\eta}.\psi) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ and a tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, in the run of the automaton for φ, a guessing or no guessing of a node labeled by x can be made. In case a guessing is made, then x can only be guessed such that it labels exactly *one* node in t whose sort is in $\widetilde{\eta}$. Such restricted guessing of x is made for the reason that x as a first order variable is used as a variable for representing a node and *not* a set of nodes. By having this restriction of guessing, we have a set of well-marked trees, where every well-marked tree in this set reflects the possibility of guessing the value of the node variable x in that tree.
  - for a formula $\varphi = (\exists X : \widetilde{\eta}.\psi) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ and a tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, in the run of the automaton for φ we could make either no guessing or some guessings for the occurrences of X in t. Compared to the first order quantification, the guessing of X can be made on several nodes in a tree t whose sorts are in $\widetilde{\eta}$, for the reason that a second order variable is used as a variable for representing a set of nodes. With this guessing mechanism, we shall still have a well-marked tree. Indeed, in a well-marked tree we have no restriction that for every second order variable, it can only be a label of exactly one node of a tree.

By having this assumption, we could then have a smaller automaton, i.e. less number of states and less number of transitions. Moreover, we also benefit from the sort information. For instance, considering a quantification formula, the sort information gives more restriction in guessing a variable on a node, since the guessing should respect both the sorts of the node and the variable.

In the following automata construction, the family $ids = \{ids_t | t \in TR(\Sigma)\}$ of functions is always given.

We construct for every $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$, such that for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, t is accepted by the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ iff t is also in $L_{\varphi}^{dec,int_{\mathcal{F}}}$. We give the idea for the construction of the automata, which also describes informally that all well-marked trees accepted by $\mathcal{M}_{dec,int_{\mathcal{F}}}$ are exactly the same as those in $L_{\varphi}^{dec,int_{\mathcal{F}}}$, and then we define the construction formally.

$\boxed{\varphi = true \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Idea

We introduce only one state, e.g. 1. The automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ runs from the leaves of the tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ upwards and always in the state 1 at any node of t until it reaches the root of t and the tree t is accepted.

Formal Construction

We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = \{1\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1, \ldots, q_k \in Q$:
  $$\delta^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = 1,$$
- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:
  $$F_{dec_{peel(t)},int_{\mathcal{F}}} = \{1\}.$$

$\boxed{\varphi = (label_\sigma(x)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

<u>Idea</u>
We introduce two states, e.g. 0 and 1. From the leaves of the tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ runs upwards and reaches the state 1, if the node $n \in V_t$ with $lab^{\Sigma}_t(n) = \sigma$ and $x \in lab^{1^{st}}_t(n)$ is visited and stays in this state until it reaches the root of t and the tree t is accepted. Otherwise, it is in state 0.

<u>Formal Construction</u>
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = \{0, 1\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1, \ldots, q_k \in Q$:
  - <u>if</u> $(\gamma = \sigma \wedge x \in U_1) \vee (\exists i \in [k] : q_i = 1)$
    <u>then</u> $\delta^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = 1$,
  - <u>otherwise</u> $\delta^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = 0$,
- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:
  $$F_{dec_{peel(t)},int_{\mathcal{F}}} = \{1\}.$$

$\boxed{\varphi = (x \in X) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

<u>Idea</u>
We introduce two states, e.g. 0 and 1. From the leaves of the tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ runs upwards and reaches the state 1, if the node $n \in V_t$ with $x \in lab^{1^{st}}_t(n)$ and $X \in lab^{2^{nd}}_t(n)$ is visited and stays in this state until it reaches the root of t and the tree t is accepted. Otherwise,

it is in state 0.

Formal Construction
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = \{0, 1\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1, \dots, q_k \in Q$:
  - if $(x \in U_1 \wedge X \in U_2) \vee (\exists i \in [k] : q_i = 1)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1, \dots, q_k) = 1$,
  - otherwise $\delta^k_{(\gamma,U_1,U_2)}(q_1, \dots, q_k) = 0$,
- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_{\mathcal{F}}} = \{1\}.$$

$\boxed{\varphi = (x == y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Idea
We introduce two states, e.g. 0 and 1. From the leaves of the tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ runs upwards and reaches the state 1, if the node $n \in V_t$ with $x \in lab^{1^{st}}_t(n)$ and $y \in lab^{1^{st}}_t(n)$ is visited and stays in this state until it reaches the root of t and the tree t is accepted. Otherwise, it is in state 0.

Formal Construction
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = \{0, 1\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1, \dots, q_k \in Q$:
  - if $(x \in U_1 \wedge y \in U_1) \vee (\exists i \in [k] : q_i = 1)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1, \dots, q_k) = 1$,
  - otherwise $\delta^k_{(\gamma,U_1,U_2)}(q_1, \dots, q_k) = 0$,
- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_{\mathcal{F}}} = \{1\}.$$

$\boxed{\varphi = (type_\eta(x)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Idea
We introduce two states, e.g. 0 and 1. From the leaves of the tree t, the automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ runs upwards and reaches the state 1, if the node $n \in V_t$ whose sort (type) is η and $x \in lab_t^{1^{st}}(n)$ is visited and stays in this state until it reaches the root of t and the tree t is accepted. Otherwise, it is in state 0.

Formal Construction
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_\mathcal{F}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = \{0, 1\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in\Sigma^{(\eta_1\ldots\eta_k,\eta_0)}_{\mathcal{V}_1,\mathcal{V}_2}}$ and $\eta_0 \ldots \eta_k \in N$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1, \ldots, q_k \in Q$:
  - if $(x \in U_1 \wedge \eta_0 = \eta) \vee (\exists i \in [k] : q_i = 1)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = 1$,
  - otherwise $\delta^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = 0$,
- $F = \{F_{dec_{peel(t)},int_\mathcal{F}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_\mathcal{F}} = \{1\}.$$

$\boxed{\varphi = (edge_i(x, y)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Idea
We introduce three states, e.g. 0, 1, and 2. From the leaves of the tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, the automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ runs upwards and reaches the state 1, if the node $n \in V_t$ with $y \in lab_t^{1^{st}}(n)$ is visited. State 2 is reached, if the node $n \in V_t$, with $x \in lab_t^{1^{st}}(n)$ and $y \in lab_t^{1^{st}}(n.i)$, is visited (if such node $n.i$ exists), propagates this state until it reaches the root of t and t is accepted. Otherwise, it is in state 0.

Formal Construction
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_\mathcal{F}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = \{0, 1, 2\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1, \ldots, q_k \in Q$:

  - Case $i \leq k$:
    if $(x \in U_1 \wedge q_i = 1) \vee (\exists j \in [k] : q_j = 2)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = 2$,
    Case $i > k$:
    if $(\exists j \in [k] : q_j = 2)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = 2$,

  - if $(y \in U_1)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = 1$,

  - otherwise $\delta^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = 0$,

- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_{\mathcal{F}}} = \{2\}.$$

$\boxed{\varphi = (x\ over\ y) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Idea
We introduce three states, e.g. 0, 1, and 2. From the leaves of the tree $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$, the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ runs upwards and reaches the state 1, if the node $n \in V_t$ with $y \in lab^{1^{st}}_t(n)$ is visited and stays in this state until the node $n \in V_t$, with $x \in lab^{1^{st}}_t(n)$ is visited and it reaches state 2, propagates this state while traversing the tree t bottom-up until it reaches the root of t and t is accepted. Otherwise, it is in state 0.

Formal Construction
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = \{0,1,2\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k \geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma,U_1,U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1,\ldots,q_k \in Q$:

  - if $(x \in U_1 \wedge \exists i \in [k] : q_i = 1) \vee (\exists i \in [k] : q_i = 2)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = 2$,

  - if $(y \in U_1) \vee (\exists i \in [k] : q_i = 1)$
    then $\delta^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = 1$,

  - otherwise $\delta^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = 0$,

- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_{\mathcal{F}}} = \{2\}.$$

$\boxed{(\varphi \in T^{Bool}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Idea
We introduce the states as l-tuples with l is the number of different variables occur in φ, where the value of the j-th component, with $j \in [l]$, of an l-tuple is either an element of $(\mathbb{N}_+ \cdot \{.\})^*$ or a dummy value.

The automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ runs upwards from the leaves of $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$ to the root of t by collecting the value of node variable x_j for every $j \in [l]$. Technically, this means we are looking for a node n in tree t which is labeled by x_j. The position of the node n in t, which is specified in Dewey notation, is then stored at the appropriate j-th component of an l-tuple. The j-th component is a dummy value only when no node, in which x_j occurs as its label, has been visited so far. At the root of t, we would have collected the value of every node variable x_j for every $j \in [l]$. The tree t is accepted whenever the state on the root of t is an l-tuple, where for every $j \in [l]$, the j-th component is the value of node variable x_j, which is a node n specified by its position in t using the Dewey notation. Moreover, the function $\tau_\theta(dec_{peel(t)}, int_\mathcal{F})(\varphi)$ should be evaluated to $True$, where $\theta(x_j) = n$ for every $j \in [l]$.

Formal Construction
Let d be a dummy value such that $d \notin (\mathbb{N}_+ \cdot \{.\})^*$. Let also $Var(\varphi)$ be the set of variables in φ and $l = card(Var(\varphi))$.

We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_\mathcal{F}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = (\{d\} \cup (\mathbb{N}_+ \cdot \{.\})^*)^l$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, and $q_1, \ldots, q_k \in Q$:

  Case: $k = 0$

  $$\delta^0_{(\gamma,U_1,U_2)}(\,) = (v_1, \ldots, v_l),$$

  where for every $j \in [l]$:

  - if $x_j \in U_1$ then $v_j = \varepsilon$,
  - otherwise $v_j = d$.

  Case: $k > 0$
  Let $q_i = (u_{i1}, \ldots, u_{il})$ for every $i \in [k]$.

  $$\delta^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = (v_1, \ldots, v_l),$$

  where for every $j \in [l]$:

  - if $x_j \in U_1$ then $v_j = \varepsilon$,

  - otherwise
    - $*$ if $(\exists! i \in [k].u_{ij} \neq d)$ then $v_j = i.u_{ij}$,
    - $*$ otherwise $v_j = d$.

- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_{\mathcal{F}}} = \{(v_1,\dots,v_l) \in ((\mathbb{N}_+ \cdot \{.\})^*)^l | \text{for every } j \in [l],\, \theta(x_j) = ids_{peel(t)}(v_j) \wedge \tau_\theta(dec_{peel(t)}, int_{\mathcal{F}})(\varphi) = True\}.$$

Note that $ids_{peel(t)}(v_j)$ for every $j \in [l]$ refers to a node $n \in V_{peel(t)}$, such that n is the value of the node variable x_j.

$\boxed{\varphi = (\neg\psi) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Let $\mathcal{M}'_{dec,int_{\mathcal{F}}} = (Q', \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta', F')$ be a deterministic bottom-up tree automaton, such that for every $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$, $t \in L(\mathcal{M}'_{dec,int_{\mathcal{F}}})$ iff $t \in L_{\psi}^{dec,int_{\mathcal{F}}}$.

Idea
The automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ is constructed from the automaton $\mathcal{M}'_{dec,int_{\mathcal{F}}}$, such that for every $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$ if t is accepted by the automaton $\mathcal{M}'_{dec,int_{\mathcal{F}}}$, then t is not accepted by the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ and, in the other way around, if t is not accepted by the automaton $\mathcal{M}'_{dec,int_{\mathcal{F}}}$, then t is accepted by the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$. This can be achieved by taking all non final states of the automaton $\mathcal{M}'_{dec,int_{\mathcal{F}}}$, regarding an interpretation function $dec_{peel(t)}$, as the final states of the automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$, regarding the same interpretation function $dec_{peel(t)}$.

Formal Construction
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = Q'$,
- $\delta = \delta'$,
- $F = \{F_{dec_t,int_{\mathcal{F}}}\}_{t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_{\mathcal{F}}} = Q' \setminus F'_{dec_{peel(t)},int_{\mathcal{F}}}.$$

$\boxed{\varphi = (\psi_1 \wedge \psi_2) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Let $\mathcal{M}^{(i)}_{dec,int_{\mathcal{F}}} = (Q^{(i)}, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta^{(i)}, F^{(i)})$ with $i \in [2]$ be a deterministic bottom-up tree automaton, such that for every $t \in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$, $t \in L(\mathcal{M}^{(i)}_{dec,int_{\mathcal{F}}})$ iff $t \in L_{\psi_i}^{dec,int_{\mathcal{F}}}$.

Idea
The automaton $\mathcal{M}_{dec,int_{\mathcal{F}}}$ is constructed from the automata $\mathcal{M}^{(1)}_{dec,int_{\mathcal{F}}}$ and

$\mathcal{M}^{(2)}_{dec,int_\mathcal{F}}$, such that for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, t is accepted by the automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ if t is accepted by both automata $\mathcal{M}^{(1)}_{dec,int_\mathcal{F}}$ and $\mathcal{M}^{(2)}_{dec,int_\mathcal{F}}$. We can view the run of the automaton for this formula as a parallel run of the automata for its subformulas. This means, we can take pairs as the states of the automaton $\mathcal{M}_{dec,int_\mathcal{F}}$, where the first component of a pair comes from $Q^{(1)}$ and the second component is from $Q^{(2)}$. The transition functions of $\mathcal{M}_{dec,int_\mathcal{F}}$ are computed by using both transition functions $\delta^{(1)}$ and $\delta^{(2)}$ and a pair of states is a final state (regarding an interpretation function $dec_{peel(t)}$) if the first component comes from $F^{(1)}_{dec_{peel(t)},int_\mathcal{F}}$ whereas the second component is from $F^{(2)}_{dec_{peel(t)},int_\mathcal{F}}$.

<u>Formal Construction</u>
We construct a deterministic bottom-up tree automaton

$$\mathcal{M}_{dec,int_\mathcal{F}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = Q^{(1)} \times Q^{(2)}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in(\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k}$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in (\Sigma_{\mathcal{V}_1,\mathcal{V}_2})_k$, $q_{11}, \ldots, q_{1k} \in Q^{(1)}$,
  and $q_{21}, \ldots, q_{2k} \in Q^{(2)}$:

$$\begin{aligned}\delta^k_{(\gamma,U_1,U_2)}((q_{11}, q_{21}), \ldots, (q_{1k}, q_{2k})) = \\ (\delta^{(1)k}_{(\gamma,U_1,U_2)}(q_{11}, \ldots, q_{1k}), \delta^{(2)k}_{(\gamma,U_1,U_2)}(q_{21}, \ldots, q_{2k}))\end{aligned}$$

- $F = \{F_{dec_{peel(t)},int_\mathcal{F}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_\mathcal{F}} = F^{(1)}_{dec_{peel(t)},int_\mathcal{F}} \times F^{(2)}_{dec_{peel(t)},int_\mathcal{F}}.$$

$\boxed{\varphi = (\exists x : \tilde{\eta}.\psi) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Let $\mathcal{M}'_{dec,int_\mathcal{F}} = (Q', \Sigma_{\mathcal{V}_1\cup\{x\},\mathcal{V}_2}, \delta', F')$ be a deterministic bottom-up tree automaton, such that for every $t' \in WTR(\Sigma, \mathcal{V}_1 \cup \{x\}, \mathcal{V}_2)$, $t' \in L(\mathcal{M}'_{dec,int_\mathcal{F}})$ iff $t' \in L^{dec,int_\mathcal{F}}_\psi$.

<u>Idea</u>
Given a tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, we guess the position of x in t. The guessing of x is made such that x is guessed to occur only on one node in t. We also restrict that we only guess on a node, whose sort is in the sort of x. We construct a nondeterministic bottom-up tree automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ from the deterministic bottom-up tree automaton $\mathcal{M}'_{dec,int_\mathcal{F}}$. The nondeterminism of the transition functions of the automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ describes whether the guessing of x is made or not. Moreover, the states of the automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ are made pairs, where the first component of a pair comes from Q' and the second component is a flag, which is used to guarantee that x is guessed to occur only on one node in t. We introduce two values of flag, i.e. $\perp$ to denote that the guess of x is not

(yet) made and $\top$ to denote that the guess of x is (already) made. Then, the tree t is accepted if a correct guessing is made, i.e. after the run of the automaton on t, we have a set of states on the root of t (since we run a nondeterministic automaton) and there is an element of this set (which is a pair, since a state for this automaton is a pair), whose first component is from $F'_{dec_{peel(t)},int_{\mathcal{F}}}$ and the second component is the flag $\top$.

Formal Construction
We construct a *nondeterministic* bottom-up tree automaton

$$\mathcal{M}_{dec,int_{\mathcal{F}}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = Q' \times \{\bot, \top\}$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in\Sigma^{(\eta_1\ldots\eta_k,\eta_0)}_{\mathcal{V}_1,\mathcal{V}_2}}$ and $\eta_0 \ldots \eta_k \in N$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in \Sigma^{(\eta_1\ldots\eta_k,\eta_0)}_{\mathcal{V}_1\cup\{x\},\mathcal{V}_2}$, and $q, q_1, \ldots, q_k \in Q'$:

  - if $\delta'^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = q$ and $x \notin U_1$
    then $\delta^k_{(\gamma,U_1,U_2)}((q_1, \bot), \ldots, (q_k, \bot)) \ni (q, \bot)$
    and for every $i \in [k]$:

    $$\delta^k_{(\gamma,U_1,U_2)}((q_1, \bot), \ldots, (q_{(i-1)}, \bot), (q_i, \top), (q_{(i+1)}, \bot), \ldots, (q_k, \bot)) \ni (q, \top)$$

  - if $\delta'^k_{(\gamma,U_1,U_2)}(q_1, \ldots, q_k) = q$ and $x \in U_1$
    then $\delta^k_{(\gamma,U_1\setminus\{x\},U_2)}((q_1, \bot), \ldots, (q_k, \bot)) \ni (q, \top)$
    (Note: this case implicitly means that we guess x only on a node whose sort is in the sort of x, since if $x \in U_1$, then the sort of the node which is labeled by (γ, U_1, U_2) must be in the sort of x, i.e. $\eta_0 \in \widetilde{\eta}$, otherwise x can not be a label of that node, i.e. $x \notin U_1$.)
- $F = \{F_{dec_{peel(t)},int_{\mathcal{F}}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

$$F_{dec_{peel(t)},int_{\mathcal{F}}} = F'_{dec_{peel(t)},int_{\mathcal{F}}} \times \{\top\}.$$

$\boxed{\varphi = (\exists X : \widetilde{\eta}.\psi) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Let $\mathcal{M}'_{dec,int_{\mathcal{F}}} = (Q', \Sigma_{\mathcal{V}_1,\mathcal{V}_2\cup\{X\}}, \delta', F')$ be a deterministic bottom-up tree automaton, such that for every $t' \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2 \cup \{X\})$, $t' \in L(\mathcal{M}'_{dec,int_{\mathcal{F}}})$ iff $t' \in L^{dec,int_{\mathcal{F}}}_{\psi}$.

Idea
Given a tree $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, we guess the position of X in t. The position of X can be guessed to occur on more than one node in t. We also restrict that we only guess on nodes, whose sorts are in the sort of x. For this purpose of guessing, we also construct a nondeterministic bottom-up tree automaton

$\mathcal{M}_{dec,int_\mathcal{F}}$ from the deterministic bottom-up tree automaton $\mathcal{M}'_{dec,int_\mathcal{F}}$. Since there is no restriction that X should be guessed to occur only on one node in t (as in guessing of first order variable), no flag is needed. The states of the automaton $\mathcal{M}_{dec,int_\mathcal{F}}$ are also the states of the automaton $\mathcal{M}'_{dec,int_\mathcal{F}}$. Then, the tree t is accepted if a correct guessing is made, i.e. after the run of the automaton on t, we have a set of states on the root of t (since we run a nondeterministic automaton) and there is an element of this set which is also contained in $F'_{dec_{peel(t)},int_\mathcal{F}}$.

<u>Formal Construction</u>
We construct a *nondeterministic* bottom-up tree automaton

$$\mathcal{M}_{dec,int_\mathcal{F}} = (Q, \Sigma_{\mathcal{V}_1,\mathcal{V}_2}, \delta, F),$$

where:

- $Q = Q'$,
- $\delta = \{\delta^k_{(\gamma,U_1,U_2)}\}_{k\geq 0,(\gamma,U_1,U_2)\in\Sigma^{(\eta_1\ldots\eta_k,\eta_0)}_{\mathcal{V}_1,\mathcal{V}_2}}$ and $\eta_0 \ldots \eta_k \in N$, where:
  for every $k \geq 0$, $(\gamma, U_1, U_2) \in \Sigma^{(\eta_1\ldots\eta_k,\eta_0)}_{\mathcal{V}_1,\mathcal{V}_2\cup\{X\}}$, and $q, q_1, \ldots, q_k \in Q'$:

  $$\underline{\text{if}}\ \delta'^k_{(\gamma,U_1,U_2)}(q_1,\ldots,q_k) = q\ \underline{\text{then}}\ \delta^k_{(\gamma,U_1,U_2\setminus\{X\})}(q_1,\ldots,q_k) \ni q$$

  (Note: this implicitly means that we guess X only on nodes whose sorts are in the sort of X, since if $X \in U_2$, then the sort of the node which is labeled by (γ, U_1, U_2) must be in the sort of X, i.e. $\eta_0 \in \widetilde{\eta}$, otherwise X can not be a label of that node, i.e. $X \notin U_2$.)
- $F = \{F_{dec_{peel(t)},int_\mathcal{F}}\}_{t\in WTR(\Sigma,\mathcal{V}_1,\mathcal{V}_2)}$, where for every $t \in WTR(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

  $$F_{dec_{peel(t)},int_\mathcal{F}} = F'_{dec_{peel(t)},int_\mathcal{F}}.$$

Chapter 7

Inductive Construction of Tree Automata

In this chapter we discuss how we construct the tree automata of MSO* formulas inductively. We shall use this inductive construction later in the implementation, where we represent a tree automaton of an MSO* formula by its transition functions and its accepting function.

To produce the transition functions of the automaton for an MSO* formula, we propose in this chapter a representation of transitions. By *transition*, we mean the application of a transition function to some given input states. We shall call our representation of a transition just as a *representation transition*. Representation transitions are needed in producing the transition functions of a formula, since we apply an inductive construction (by the structure of the formula) for producing the transition functions. Such inductive construction will often produce intermediate results. Thus, there is a need to represent the transitions such that the number of the representation transitions is less compared to the transitions which are produced in the naive construction, where all possible input symbols and states are considered. At the end of the inductive construction, we obtain the final representation transitions which are later unfolded to derive the transition functions.

The aim to gain less number of representation transitions can be achieved in several ways: by representing the transitions which have the same behavior as one representation transition, by considering the sort information, by considering only the reachable states, and by considering our assumption that an automaton runs on well-marked trees.

To produce an accepting function of the automaton for an MSO* formula, we define it also inductively on the structure of the formula. But, in contrast to the transition functions, we do not need a special representation for the accepting functions. The reason is that the accepting function is used when the state which is reached after the automaton has run on a tree is already obtained. By keeping the structure of the states (in particular also for the state reached on the root of a tree), it can be decided whether a tree is accepted or not. This can be done by applying the definition of the accepting function inductively on the structure of the state (which have the same structure as the formula).

To be able to represent the transitions, we shall give first the representation of the extended alphabets we defined in the definition 24. We also give a representation of the states of the automaton for every MSO* formula.

For the examples we give throughout this chapter, we consider abstract syntax trees which represent the (nonempty) declaration part of the programming language *Simple* from chapter 1. This means, the labels for the nodes of such abstract syntax trees consist only of the following:

- nullary symbols: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, $\texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}$, $\texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}$,
- unary symbol: $\texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}$,
- binary symbols: $\texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}$, $\texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}$.

We introduce three node variables:

- x, whose possible value is a node with the sort `Decl`,
- y, whose possible value is a node with the sort `Ident`,
- z, whose the possible value is a node with the sort `Ident`.

We also introduce two node set variables:

- X, whose possible value is a set of nodes with the sort `Ident`,
- Y, whose possible value is a set of nodes with the sort `TypExp`.

The nodes which have the sort `Ident` are provided with the attribute *name*, whose value is the name of the corresponding identifier. And the nodes which have the sort `Decl` and `DList` are provided with the attribute $often$ which has one context argument. The value of the attribute $often$ on a node n (with the sort `Decl` or `DList`) describes the number of its indirect successor nodes with the sort `Ident`, where the attribute *name* for each of these indirect successor nodes has the value as given in the context argument of the attribute $often$ for the node n. Additionally, we also define two functions, viz. the nullary function *null* which is interpreted as the natural number 0 and the binary function *gt* which is no other than the binary comparison operator '>' over natural numbers. With this attributes and functions, we can ask for example, whether in a declaration part a particular variable is declared or not.

Based on this motivation, we define N, Σ, $\mathcal{V}_1$, $\mathcal{V}_2$, K, $\mathcal{A}$, and $\mathcal{F}$ for our examples throughout this chapter in table 7.1.

7.1 Representation of The Extended Alphabets

Definition 28 (Representation of The Extended Alphabets). Let $\overline{\mathcal{V}_1} = \{\overline{x}^{\widetilde{\eta}} \mid x^{\widetilde{\eta}} \in \mathcal{V}_1 \wedge \widetilde{\eta} \in \mathcal{P}(N)\}$ be a $\mathcal{P}(N)$-sorted set and $\widehat{\mathcal{V}_1} = \mathcal{V}_1 \cup \overline{\mathcal{V}_1}$ be a $\mathcal{P}(N)$-sorted set. Let also $\overline{\mathcal{V}_2} = \{\overline{X}^{\widetilde{\eta}} \mid X^{\widetilde{\eta}} \in \mathcal{V}_2 \text{ and } \widetilde{\eta} \in \mathcal{P}(N)\}$ be a $\mathcal{P}(N)$-sorted set and $\widehat{\mathcal{V}_2} = \mathcal{V}_2 \cup \overline{\mathcal{V}_2}$ be a $\mathcal{P}(N)$-sorted set.

We represent our extended alphabet $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$ as the following set:

$$\begin{aligned} \mathit{Ext}(\Sigma,\mathcal{V}_1,\mathcal{V}_2) &= \{(E,E_1,E_2) \mid E \subseteq \Sigma,\ E_1 \subseteq \widehat{\mathcal{V}_1},\ E_2 \subseteq \widehat{\mathcal{V}_2}, \\ &\quad \forall\sigma \in E.\forall\varkappa \in E_1.\ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(\varkappa), \\ &\quad \forall\sigma \in E.\forall\varkappa \in E_2.\ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(\varkappa)\}. \end{aligned}$$

The elements of $\mathit{Ext}(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$ are called *representation input symbols.* □

By having this representation of the extended alphabets, one representation input symbol may cover more than one input symbol from $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$. To be more precise, we give the interpretation of a representation input symbol $(E,E_1,E_2) \in \mathit{Ext}(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$:

Given a representation input symbol $(E,E_1,E_2) \in \mathit{Ext}(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$, this symbol covers every symbol $(\sigma,U_1,U_2) \in \Sigma_{\mathcal{V}_1,\mathcal{V}_2}$, where:

- $\sigma \in E$,
- U_1 always contains all variables represented without overlines in E_1, but does not contain any variable represented using overlines in E_1. Additionally, U_1 may also contain other variables from $\mathcal{V}_1$ (which is not in E_1) by considering appropriate sort with respect to definition 24,
- U_2 always contains all variables represented without overlines in E_2, but does not contain any variable represented using overlines in E_2. Additionally, U_2 may also contain other variables

- $N = \{\texttt{Ident}, \texttt{TypExp}, \texttt{Decl}, \texttt{DList}\}$
- $\Sigma = \{\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}, \texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}, \texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}, \texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}, \texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}, \texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}\}$
- $\mathcal{V}_1 = \{x^{\{\texttt{Decl}\}}, y^{\{\texttt{Ident}\}}, z^{\{\texttt{Ident}\}}\}$
- $\mathcal{V}_2 = \{X^{\{\texttt{Ident}\}}, Y^{\{\texttt{TypExp}\}}\}$
- $K = \{\texttt{String}, \texttt{Int}, \texttt{Bool}\}$
- $\mathcal{A} = \{\mathit{name}^{(\{\texttt{Ident}\},\varepsilon,\texttt{String})}, \mathit{often}^{(\{\texttt{Decl},\texttt{DList}\},\texttt{String},\texttt{Int})}\}$, with:
  For every $t \in \mathit{WTR}(\Sigma,\mathcal{V}_1,\mathcal{V}_2)$:
  - for every $\langle \mathit{name}, n\rangle \in \mathcal{I}\langle\mathcal{A},V_t\rangle^{(\varepsilon,\texttt{String})}$, $\mathit{dec}_{\mathit{peel}(t)}(\langle \mathit{name}, n\rangle)$ is the name of the identifier corresponding to the node n.
  - for every $\langle \mathit{often}, n\rangle \in \mathcal{I}\langle\mathcal{A},V_t\rangle^{(\texttt{String},\texttt{Int})}$ and $\mathit{val} \in \Omega^{\texttt{String}}$, $\mathit{dec}_{\mathit{peel}(t)}(\langle \mathit{often}, n\rangle)(\mathit{val})$ is the number of the indirect successor nodes of n with the sort `Ident`, where the attribute *name* for each of these indirect successor nodes has the value *val*.
- $\mathcal{F} = \{\mathit{null}^{(\varepsilon,\texttt{Int})}, \mathit{gt}^{(\texttt{Int Int},\texttt{Bool})}\}$, with:
  $\mathit{int}_{\mathcal{F}}(\mathit{null})() = 0$ and for every $e_1, e_2 \in \texttt{Int}$, $\mathit{int}_{\mathcal{F}}(\mathit{gt})(e_1,e_2) = e_1 > e_2$ holds.

Table 7.1: N, Σ, $\mathcal{V}_1$, $\mathcal{V}_2$, K, $\mathcal{A}$, and $\mathcal{F}$ for examples throughout this chapter

from $\mathcal{V}_2$ (which is not in E_2) by considering appropriate sort with respect to definition 24.

In particular, if $E_1 = \{\}$ (and $E_2 = \{\}$, respectively), then U_1 (and U_2, respectively) may contain any variable from $\mathcal{V}_1$ (and from $\mathcal{V}_2$, respectively) by considering appropriate sort with respect to definition 24.

In our representation of the extended alphabets, we especially introduce a representation variable with overline for the following purposes:

- Our representation input symbol should be able to describe the following different situations:
  - A node variable (or a node set variable, respectively) is a part of an input symbol.
  - A node variable (or a node set variable, respectively) is *not* a part of an input symbol.
  - A node variable (or a node set variable) is either a part or *not* a part of an input symbol. This is the case, if the occurrence of a variable as a part of an input symbol is not significant for the behavior of transition functions with that input symbol. For example, if we consider the transition function of the automaton for the formula $label_\sigma(x)$, then the transition functions of every input symbol without σ as its part (independent of the fact that x is also a part of that input symbol or not) will give the same output states.

  For the first case, we represent it by writing explicitly the variable without overline as a part of the representation input symbol. In the second case, we give the representation by writing explicitly the variable using overline as a part of the representation input symbol, whereas for the third case we represent it by not writing the variable at all. In this way, these three situations can be distinguished.

- In our representation of transition functions, it gives an additional information for a representation input symbol to restrict that a node variable can only be a label of exactly one node in a tree. For example, if we consider the transition function of the automaton for the formula $label_\sigma(x)$, where the input states contain the state 1 (which means that we have seen a node labeled by σ and x), then we can use $\overline{x}$ as part of the representation input symbol in our representation transition function to describe this situation.

- Considering the cross product construction for our representation transitions, having a representation variable with overline, e.g $\overline{x}$, is useful to reduce the number of transition functions by leaving out unnecessary representation transition functions (those which will never be used due to our assumption). This particularly happens, when we deal with two formulas having x occurs in both formulas, where we can avoid producing a new transition functions from these two formulas, if in one transition function $\overline{x}$ occurs in the representation input symbol (which means that x should not be a label) whereas in the other transition function x occurs in the representation input symbol (which means that x should be a label). A similar situation can also happen when we consider the projection and the power set constructions for our representation. We shall see this more clearly when we discuss the constructions.

We give some examples of representation input symbols.

Example 1. Consider the following representation input symbols:

- $(\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ represents the following symbols from $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$:
  - $(\texttt{INTTYP}, \{\}, \{\})$,
  - $(\texttt{INTTYP}, \{\}, \{Y\})$,
  - $(\texttt{BOOLTYP}, \{\}, \{\})$,
  - $(\texttt{BOOLTYP}, \{\}, \{Y\})$.
- $(\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ represents the following symbols from $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$:
  - $(\texttt{IDENT}, \{\}, \{\})$,
  - $(\texttt{IDENT}, \{\}, \{X\})$,
  - $(\texttt{IDENT}, \{z\}, \{\})$,
  - $(\texttt{IDENT}, \{z\}, \{X\})$.

□

7.2 Representation of The States

In this section we give a representation of states of the automaton for an MSO* formula. Except for the attribute formula, the states of the automata for the other atomic formulas do not have any special representation, since the number of states of these automata is finite (with at most only three states). Hence, for their representation we can use the concrete values.

On the other hand, we need to give a special representation of states of the automaton for an attribute formula, since the number of states for this formula is infinite. Therefore, we propose a symbolic representation for representing the concrete values of the components of the states for this formula. We propose the following symbolic representations:

1. The symbolic representation of *a node*. This symbolic representation is used to represent the path information (a dummy value or a Dewey notation) of a node only. It consists of the following:
   - d_x, where $x \in \mathcal{V}_1$, is used to represent a dummy value (no path information) for the node variable x which occurs in an attribute formula,
   - nd_x, where $x \in \mathcal{V}_1$, is a general representation for all Dewey notation. It represents the path information of a node, where this node is the value of the node variable x which occurs in an attribute formula.

- ε_x, where $x \in \mathcal{V}_1$, is a particular representation for the Dewey notation ε. The given index x has a similar meaning as in nd_x.
- $i.nd_x$, where $i \in \mathbb{N}_+$ and $x \in \mathcal{V}_1$, is a particular representation for a Dewey notation which has a prefix i. The given index x also has a similar meaning as in nd_x.

2. The symbolic representation of *a set of nodes.* It is used to represent the path information for a set of nodes. This symbolic representation is needed when we deal later with the power set construction. As we know from 5.3, this construction is used to determinize a nondeterministic automaton, which we obtain if we transform a formula using quantifier. In particular, if we deal with a first order quantification formula and we quantify a node variable which occurs in an attribute formula, then we may have some guessed nodes as the value of the quantified node variable. To represent the path information of these guessed nodes, instead of using a symbolic representation of a node, we use a symbolic representation of a set of nodes to represent the path information of those guessed nodes. The symbolic representation consists of the following:

   - $\emptyset_x$, where $x \in \mathcal{V}_1$, is used to represent a set containing no path information, since there are no guessed nodes for the node variable x.
   - $snd_x^{(p,q)}$, where $x \in \mathcal{V}_1$ and $p, q \in \mathbb{N}_+$, is used to represent a set containing the path information of guessed nodes for the node variable x. We give an additional superscript index (p, q), since when we deal with the power set construction, we may have several symbolic representations of a set of nodes for the node variable x in the input states of a representation transition. This can happen for the following reasons:
     - The symbolic representation for the path information of the nodes which are guessed from different branches should be distinguishable, since different branches will give different prefix to the path information of the guessed nodes collected so far. For this reason, we distinguish the symbolic representation by the first index p.
     - If a symbolic representation is a part of a more complex structure of state (in case an attribute formula is a subformula of a more complex formula), then this symbolic representation should be distinguishable in every different state. The reason is that the symbolic representations for the same node variable which occur in different states represent different path information of the guessed nodes. For this reason, we give the second index q.
   - $sc\varepsilon_x$, where $x \in \mathcal{V}_1$, is used to represent a singleton set containing only ε as the path information of the node which is guessed for the node variable x. It is not necessary to have an additional superscript index, since this symbolic representation is reached only when we have no guessed node for the node variable x so far, which is represented by $\emptyset_x$.

- $i.snd_x^{(p,q)}$, where $x \in \mathcal{V}_1$ and $i, p, q \in \mathbb{N}_+$, is used to represent a set containing the path information of some guessed nodes for the node variable x, where the path information of every guessed node is built by giving the prefix i to the path information of every node represented in $snd_x^{(p,q)}$. The additional superscript index (p, q) are added with the same reasons as in $snd_x^{(p,q)}$.
- $sms_x(s_1, s_2)$, where $x \in \mathcal{V}_1$ and both s_1 and s_2 are symbolic representations of a set of nodes. This symbolic representation is used to represent a set containing the path information of the guessed nodes for x represented in s_1 and also the path information of the guessed nodes for x represented in s_2. The purpose of introducing this symbolic representation is to collect the path information of all guessed nodes for x we have so far.

As we shall see later, our power set construction is based on a reachability construction, where only representation transitions for the reachable states are computed. If a symbolic representation of a set of nodes is a part of a reachable state, it turns out that the additional superscript plays no role. The additional superscript index is only needed whenever the reachable state becomes an input state of a representation transition. For this purpose of reachable states, additionally we also introduce snd_x (without an additional superscript index) as a symbolic representation of a set of nodes, which can be seen as a general representation of a set of nodes which are guessed for the node variable x.

In the following we give the formal definition for the representation of the states of the automaton for every MSO* formula.

Definition 29 (Representation of The States). Let $Dw_{\mathcal{V}_1} = \{d_x \mid x \in \mathcal{V}_1\} \cup \{nd_x \mid x \in \mathcal{V}_1\} \cup \{\varepsilon_x \mid x \in \mathcal{V}_1\} \cup \{i.nd_x \mid i \in \mathbb{N}_+,\ x \in \mathcal{V}_1\}$ be the set of symbolic representations for the Dewey notation (path information) of a node and $Sdw_{\mathcal{V}_1} = \{\emptyset_x \mid x \in \mathcal{V}_1\} \cup \{snd_x^{(p,q)} \mid x \in \mathcal{V}_1,\ p, q \in \mathbb{N}_+\} \cup \{sc\varepsilon_x \mid x \in \mathcal{V}_1\} \cup \{i.snd_x^{(p,q)} \mid x \in \mathcal{V}_1,\ i, p, q \in \mathbb{N}_+\} \cup \{sms_x(s_1, s_2) \mid x \in \mathcal{V}_1,\ s_1, s_2 \in Sdw_{\mathcal{V}_1}\} \cup \{snd_x \mid x \in \mathcal{V}_1\}$ be the set of symbolic representations for the Dewey notation (path information) of a set of nodes. Let also $Gs = \{\bot, \top\}$ be the set of flags used for the guessing mechanism in the construction of the automaton for a first order quantification formula.

Let $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. We define a family $\mathcal{S}t(\varphi) = \{\mathcal{S}t(\varphi)_{U_1} | U_1 \subseteq \mathcal{V}_1\}$ of sets of representation of states for the deterministic automaton of φ considering U_1. The set U_1 contains the node variables, which are bounded in the formula where φ is its subformula, but does not contain the node variables, which are bounded in φ. Similarly, we also define $\mathcal{S}t_{non}(\varphi) = \{\mathcal{S}t_{non}(\varphi)_{U_1} | U_1 \subseteq \mathcal{V}_1\}$ for the nondeterministic automaton of φ.

The set of representation of states for the deterministic automaton of φ is defined as the following:

$$St(\varphi) = \mathcal{S}t(\varphi)_\emptyset,$$

and the set of representation of states for the nondeterministic automaton of φ

is defined as the following:

$$St_{non}(\varphi) = \mathcal{S}t_{non}(\varphi)_{\emptyset}.$$

We define for every $U_1 \subseteq \mathcal{V}_1$, $\mathcal{S}t(\varphi)_{U_1}$ as the following:

- $\varphi = true$
  $\mathcal{S}t(\varphi)_{U_1} = \{1\}$.
- $\varphi = label_\sigma(x)$
  $\mathcal{S}t(\varphi)_{U_1} = \{0, 1\}$.
- $\varphi = x \in X$
  $\mathcal{S}t(\varphi)_{U_1} = \{0, 1\}$.
- $\varphi = (x == y)$
  $\mathcal{S}t(\varphi)_{U_1} = \{0, 1\}$.
- $\varphi = type_\eta(x)$
  $\mathcal{S}t(\varphi)_{U_1} = \{0, 1\}$.
- $\varphi = edge_i(x, y)$
  $\mathcal{S}t(\varphi)_{U_1} = \{0, 1, 2\}$.
- $\varphi = x\ over\ y$
  $\mathcal{S}t(\varphi)_{U_1} = \{0, 1, 2\}$.
- $\varphi \in T^{Bool}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}$

$$\begin{aligned}\mathcal{S}t(\varphi)_{U_1} \quad = \quad & \{(v_1, \ldots, v_l) \mid l \text{ is the number of different variables in } \varphi, \\ & \text{and for every } j \in [l], \underline{\text{if}}\ x_j \in U_1, \underline{\text{then}}\ v_j \in Sdw_{\mathcal{V}_1}, \\ & \underline{\text{otherwise}}\ v_j \in Dw_{\mathcal{V}_1}\}\end{aligned}$$

- $\varphi = \neg\psi$
  $\mathcal{S}t(\varphi)_{U_1} = \mathcal{S}t(\psi)_{U_1}$.
- $\varphi = \psi_1 \wedge \psi_2$
  $\mathcal{S}t(\varphi)_{U_1} = \{(q_1, q_2) \mid q_1 \in \mathcal{S}t(\psi_1)_{U_1},\ q_2 \in \mathcal{S}t(\psi_2)_{U_1}\}$.
- $\varphi = \exists x : \widetilde{\eta}.\ \psi$
  $\mathcal{S}t(\varphi)_{U_1} = \mathcal{P}(\mathcal{S}t_{non}(\varphi)_{U_1})$, where:
  $\mathcal{S}t_{non}(\varphi)_{U_1} = \{(q, f) \mid q \in \mathcal{S}t(\psi)_{U_1\cup\{x\}},\ f \in Gs\}$.
- $\varphi = \exists X : \widetilde{\eta}.\ \psi$.
  $\mathcal{S}t(\varphi)_{U_1} = \mathcal{P}(\mathcal{S}t_{non}(\varphi)_{U_1})$, where:
  $\mathcal{S}t_{non}(\varphi)_{U_1} = \mathcal{S}t(\psi)_{U_1}$

$\square$

7.3 Representation of The Transitions

By using our representation of the extended alphabets and the states, in the following we give a representation of the transitions for the automata of atomic formulas and show how the cross product, projection, and the power set constructions are realized by using this representation.

7.3.1 Representation Transitions of Atomic Formulas

Since we attempt to have less number of representation transitions (compared to the number of transitions using the naive construction), in general we come to the following idea for representing the transitions of the atomic formulas:

- We represent some transitions which have the same behavior as one representation transition. For example, if we consider the formula $label_{\texttt{DECL}}(x)$, then the transitions for input symbols whose first components are other than `DECL`, e.g. the elements of Σ whose ranks are 0: `IDENT`, `INTTYP`, and `BOOLTYP` (regardless of the first order and second order variables in their second and third components), will have the same output state, i.e. the state 0. In this case, we benefit from our representation of the extended alphabets, where we can combine those transitions by combining their input symbols. This means, instead of having the transitions with the following input symbols (for the rank 0):

  - $(\texttt{IDENT}, \{\}, \{\})$, $(\texttt{IDENT}, \{\}, \{X\})$, $(\texttt{IDENT}, \{y\}, \{\})$, $(\texttt{IDENT}, \{y\}, \{X\})$, $(\texttt{IDENT}, \{z\}, \{\})$, $(\texttt{IDENT}, \{z\}, \{X\})$, $(\texttt{IDENT}, \{y, z\}, \{\})$, $(\texttt{IDENT}, \{y, z\}, \{X\})$,
  - $(\texttt{INTTYP}, \{\}, \{\})$, $(\texttt{INTTYP}, \{\}, \{Y\})$,
  - $(\texttt{BOOLTYP}, \{\}, \{\})$, $(\texttt{BOOLTYP}, \{\}, \{Y\})$,

  we can have only one representation transition function using a representation input symbol, namely $(\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\})$, which covers all the above mentioned input symbols.

- We do not represent the transitions which are never used. Such transitions can be left out in our representation for several reasons:

  - Since we have the assumption that an automaton runs on well-marked trees, there are transitions whose input states are not reachable by this assumption. Such transitions are not necessary to be represented. For instance, if we reconsider the formula $label_{\texttt{DECL}}(x)$, then we do not need to represent the transitions in which the state 1 occurs more than once in their input states (since state 1 means that we have seen a node labeled by `DECL` and x and by our assumption, x can not label more than one node).
  - Since we consider the ranked alphabets, the variables, and the nodes are sorted, there are transitions whose input states reflect a labeling of a variable at a node without respecting the sort. Such transitions are also not necessary to be represented. For instance, if we consider the formula $edge_1(x, y)$, then we do not need to represent the transition with the input symbol $(\texttt{DPAIR}, \{x\}, \{\})$ and the input states $(1, 0)$, since this input states means that the first successor of the node labeled by $(\texttt{DPAIR}, \{x\}, \{\})$ is labeled by the node variable y (which is reflected by the state 1 in the first input state). But, we would not have such situation, since the sort of y is $\{\texttt{Ident}\}$, and the first successor of the node labeled by $(\texttt{DPAIR}, \{x\}, \{\})$ is the node which has the sort `Decl`.

The representation should not only aim to minimize the number of transitions, but moreover the transitions are represented in the following way:

- *Complete*, that means, with respect to our assumption, given an input symbol and a tuple of input states, we can always find a representation of transition for that input symbol and tuple of input states. This can be achieved if we consider all possible representation input symbols for every possible configuration of input states with respect to our assumption.
- *Non-overlapping*, that means, with respect to our assumption, given an input symbol and a tuple of input states, we have exactly one transition in our representation for that input symbol and tuple of states. This can be achieved if we define all possible representation input symbols non-overlappingly for every possible configuration of input states with respect to our assumption.

Based on these general ideas, we can now define the representation transitions for atomic formulas. For every formula, we start by giving the informal description of how we define the representation transitions and then we give the formal definition.

First of all, we need to define the function $dtrans : MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}} \rightarrow \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$, which takes a formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ and gives a set $dtrans(\varphi) \in \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$ of representation transitions for the deterministic bottom-up tree automaton of the formula φ, where $Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ is a set of representation transitions, such that for every (rank) $k \in \mathbb{N}$, $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, and $dt_{k,\varphi} \in Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

$$dt_{k,\varphi} : Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi)^k \rightarrow St(\varphi).$$

If φ is clear from the context, we write dt_k instead of $dt_{k,\varphi}$.

We define the representation transitions for every atomic formula as the following:

$\boxed{\varphi = true \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

Since the automaton for this formula has only one state, i.e. state 1, then for every rank $k \in Rank_\Sigma$, we have only one possible k-tuple of input states, i.e. all input states in this k-tuple are 1. For this tuple of input states, we have only one transition for every $k \in Rank_\Sigma$, that covers every input symbol in $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$ with the rank k. In our representation, those symbols can be represented by $(\Sigma_k, \{\}, \{\})$.

Formally, we define $dtrans(\varphi)$ as the smallest set by the following scheme:

For every $k \in Rank_\Sigma$:

$$(dt_k((\Sigma_k, \{\}, \{\}), (q_1, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

with $q_i = 1$ for every $i \in [k]$.

Since the representation input symbol $(\Sigma_k, \{\}, \{\})$ covers every input symbol in $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$ with the rank k and we define a representation of transition using our representation input symbol for every rank k, then the transition functions for this formula are represented completely and non-overlappingly in this way.

We give an example.

Example 2. Consider the formula $\varphi = \mathit{true} \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Then:

$$\begin{aligned} \mathit{dtrans}(\varphi) \quad = \{ \quad & \mathit{dt}_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 1, \\ & \mathit{dt}_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1, \\ & \mathit{dt}_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (1,1)) = 1 \quad \} \end{aligned}$$

We can compare this result with the transitions which are constructed by following the formal construction given in subsection 6.2.2, where we consider for every input symbol, every state as the input of the transition functions. By representing the transitions in this way, we obtain only 3 representation transitions, whereas following the formal construction we obtain 16 transitions:

- 8 transitions having the input symbols with the first component: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, where we consider 2 node variables ($y^{\{\texttt{Ident}\}}$ and $z^{\{\texttt{Ident}\}}$), 1 node set variable ($X^{\{\texttt{Ident}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 1 transition having the input symbol with the first component: $\texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}$, where we consider no node variable, no node set variable, and one combination of input states, i.e. (1),
- 1 transition having the input symbol with the first component: $\texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}$, where we consider no node variable, no node set variable, and one combination of input states, i.e. (1,1).
- 2 transitions having the input symbols with the first component: $\texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}$, where we consider 1 node variable ($x^{\{\texttt{Decl}\}}$), no node set variable, and one combination of input states, i.e. (1,1). $\square$

We handle the following four atomic formulas similarly:

- $\varphi = \mathit{label}_\sigma(x) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$
- $\varphi = (x \in X) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$
- $\varphi = (x == y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$
- $\varphi = \mathit{type}_\eta(x) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$

where we define, for every rank $k \in \mathit{Rank}_\Sigma$, the representation transitions with:

- all components of their k-tuples of input states are the state 0,
- exactly one component of their k-tuples of input states is the state 1, whereas the other components are the states 0. Recall from subsection 6.2.2 that the state 1 is only reached:
  - if we have visited a node labeled by x and σ, for the formula $label_\sigma(x)$,
  - if we have visited a node labeled by x and X, for the formula $x \in X$,
  - if we have visited a node labeled by x and y, for the formula $x == y$,
  - if we have visited a node labeled by x and η is the sort of the node, for the formula $type_\eta(x)$.

  Since we consider well-marked trees on which the automata run, then there exists exactly one node which is labeled by x. This means we do not need to represent the transitions with more than one component of their k-tuples of input states are the state 1, since those transitions will never be used due to our assumption.

These are all the possible input states of the representation transitions we may have for these four atomic formulas with respect to our assumption. By considering these possible input states, in the following we discuss how we define the representation of the transitions for every of these four formulas. In particular, we show how we group the input symbols from the extended ranked alphabet (considering the behavior of the transitions) by means of our representation input symbols, such that the transitions are defined completely and non-overlappingly.

$\boxed{\varphi = (label_\sigma(x)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

For this formula, first we consider the input symbols which have the same ranks as the rank of σ and then we consider the other ranks:

1. For symbols of Σ_k, where $k \in Rank_\Sigma$ is the rank of σ:
   (a) For k-tuple of input states where all components are the state 0, we define the representation transitions for the following representation input symbols:
   - $(\{\sigma\}, \{x\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0,
   - $(\Sigma_k \setminus \{\sigma\}, \{\}, \{\})$ – if the input symbols represented by this representation exist – where the output state of the representation transition for this representation input symbol is 0.

   These are all the possible representation input symbols which are complete and non-overlapping for the k-tuple of input states where all components are the state 0.

   (b) For k-tuple of input states with exactly one component is the state 1 (and the other components are the state 0), we define the representation transitions for the following representation input symbols (if they exist):

- $(\{\alpha \in \Sigma_k \mid sort_{res}(\alpha) \in sort(x)\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
- $(\{\alpha \in \Sigma_k \mid sort_{res}(\alpha) \notin sort(x)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1.

These are all the possible representation input symbols which are complete and non-overlapping, with respect to our assumption, for the k-tuple of input states with exactly one component is the state 1. Note that for the first item, it is not necessary to define the representation transition for the representation input symbol $(\{\alpha \in \Sigma_k \mid sort_{res}(\alpha) \in sort(x)\}, \{x\}, \{\})$, since for this k-tuple of input states it will never be used (recall that the state 1 in this k-tuple of input states means we have visited a node in a well-marked tree which is labeled by σ and x, hence there are no more nodes in that well-marked tree which can be labeled by x).

2. For symbols of $\Sigma_{k'}$, for every $k' \in Rank_\Sigma \setminus \{rank(\sigma)\}$:

   (a) For k'-tuple of input states where all components are the state 0, we define the representation transitions for the representation input symbol $(\Sigma_{k'}, \{\}, \{\})$, where the output of the representation transition for this representation input symbol is 0.

   Since the representation input symbol $(\Sigma_{k'}, \{\}, \{\})$ covers every input symbol in $\Sigma_{\mathcal{V}_1,\mathcal{V}_2}$ with the rank k', then for this k'-tuple of input states we have only one representation transition. In this way, the transitions for this k'-tuple of input states are represented completely and non-overlappingly.

   (b) For k'-tuple of input states with exactly one component is state 1 (and the other components are the state 0), we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\alpha \in \Sigma_{k'} \mid sort_{res}(\alpha) \in sort(x)\}, \{\overline{x}\}, \{\})$, where the output of the representation transition for this representation input symbol is 1,
   - $(\{\alpha \in \Sigma_{k'} \mid sort_{res}(\alpha) \notin sort(x)\}, \{\}, \{\})$, where the output of the representation transition for this representation input symbol is 1.

   These are all the possible representation input symbols which are complete and non-overlapping, with respect to our assumption, for the k-tuple of input states with exactly one component is the state 1. Note that for the first item, it is also not necessary to define the representation transitions for the representation input symbol $(\{\alpha \in \Sigma_k \mid sort_{res}(\alpha) \in sort(x)\}, \{x\}, \{\})$ for the same reason as mentioned in item 1b.

Formally, we define $dtrans(\varphi)$ as the smallest set by the following scheme:

If $rank(\sigma) = k$ with $k \in Rank_\Sigma$, then:

(considering the k-tuple of input states with all components are the state 0)

- $(dt_k((\{\sigma\}, \{x\}, \{\}), (q_1, \ldots, q_k)) = 1) \in dtrans(\varphi)$ with $q_i = 0$ for every $i \in [k]$,
- $(dt_k((\{\sigma\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_k)) = 0) \in dtrans(\varphi)$ with $q_i = 0$ for every $i \in [k]$,
- if $\Sigma_k \setminus \{\sigma\} \neq \emptyset$, then:
  $(dt_k((\Sigma_k \setminus \{\sigma\}, \{\}, \{\}), (q_1, \ldots, q_k)) = 0) \in dtrans(\varphi)$ with $q_i = 0$ for every $i \in [k]$,

(considering the k-tuple of input states with exactly one component is the state 1)

- if there is $\alpha \in \Sigma_k$ with $sort_{res}(\alpha) \in sort(x)$, then:
  for every $i \in [k]$ with $q_i = 1$:

  $$(dt_k((\{\alpha \in \Sigma_k \mid sort_{res}(\alpha) \in sort(x)\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $j \in [k] \setminus \{i\}$: $q_j = 0$,
- if there is $\alpha \in \Sigma_k$ with $sort_{res}(\alpha) \notin sort(x)$, then:
  for every $i \in [k]$ with $q_i = 1$:

  $$(dt_k((\{\alpha \in \Sigma_k \mid sort_{res}(\alpha) \notin sort(x)\}, \{\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $j \in [k] \setminus \{i\}$: $q_j = 0$.

And for every $k' \in Rank_\Sigma \setminus \{rank(\sigma)\}$, we have the following:

(considering the k-tuple of input states with all components are the state 0)

- $(dt_{k'}((\Sigma_{k'}, \{\}, \{\}), (q_1, \ldots, q_{k'})) = 0) \in dtrans(\varphi)$ with $q_i = 0$ for every $i \in [k']$,

(considering the k-tuple of input states with exactly one component is the state 1)

- if there is $\alpha \in \Sigma_{k'}$ with $sort_{res}(\alpha) \in sort(x)$, then:
  for every $i \in [k']$ with $q_i = 1$:

  $$(dt_{k'}((\{\alpha \in \Sigma_{k'} \mid sort_{res}(\alpha) \in sort(x)\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_i, \ldots, q_{k'})) = 1) \in dtrans(\varphi),$$

  where for every $j \in [k'] \setminus \{i\}$: $q_j = 0$,

- <u>if</u> there is $\alpha \in \Sigma_{k'}$ with $sort_{res}(\alpha) \notin sort(x)$, <u>then</u>:
  for every $i \in [k']$ with $q_i = 1$:

  $$(dt_{k'}((\{\alpha \in \Sigma_{k'} \mid sort_{res}(\alpha) \notin sort(x)\}, \{\}, \{\}), (q_1, \dots, q_i, \dots, q_{k'})) = 1) \in dtrans(\varphi),$$

  where for every $j \in [k'] \setminus \{i\}$: $q_j = 0$.

We give an example.

Example 3. Consider the formula $\varphi = label_{\texttt{DECL}}(x) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Then:

$$\begin{aligned} dtrans(\varphi) = \{\ & dt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0, \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0, \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1, \\ & dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (0,0)) = 1, \\ & dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0,0)) = 0, \\ & dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (1,0)) = 1, \\ & dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0,1)) = 1, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,0)) = 0, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (1,0)) = 1, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,1)) = 1 \quad \} \end{aligned}$$

□

$\boxed{\varphi = (x \in X) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

For this formula, we consider for every rank $k \in Rank_\Sigma$ the following:

1. For k-tuple of input states with all components are the state 0, we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x\}, \{X\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x\}, \{\overline{X}\})$, where the output state of the representation transition for this representation input symbol is 0,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\ \})$, where the output state of the representation transition for this representation input symbol is 0,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0.

   These are all the possible representation input symbols which are complete and non-overlapping for the k-tuple of input states where all components are the state 0. Note that we choose to group the input symbols for this k-tuple of input states by deciding for every $\sigma \in \Sigma_k$, whether $sort_{res}(\sigma) \in sort(x)$ or $sort_{res}(\sigma) \notin sort(x)$. Another alternative is to group them by deciding for every $\sigma \in \Sigma_k$, whether $sort_{res}(\sigma) \in sort(X)$ or $sort_{res}(\sigma) \notin sort(X)$, but this makes no difference, since for this formula $sort(x) = sort(X)$ (cf. definition 23).

2. For k-tuple of input states with exactly one component is the state 1 (and the other components are the state 0), we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1.

   These are all the possible representation input symbols which are complete and non-overlapping, with respect to our assumption, for the k-tuple of input states with exactly one component is the state 1. Note that for the first item, it is not necessary to define the representation transition for the representation input symbol $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x\}, \{\})$, since for this k-tuple of input states it will never be used (recall that the state 1 in this k-tuple of input states means that we have visited a node in a well-marked tree which is labeled by x and X, hence there are no more nodes in that well-marked tree that can be labeled by x).

Formally, we define $dtrans(\varphi)$ as the smallest set by the following scheme:

For every $k \in Rank_{\Sigma}$:

(considering the k-tuple of input states with all components are the state 0)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ <u>then</u>:
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x\}, \{X\}), (q_1, \ldots, q_k)) = 1)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x\}, \{\overline{X}\}), (q_1, \ldots, q_k)) = 0)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_k)) = 0)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ <u>then</u>:

  $$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\}), (q_1, \ldots, q_k)) = 0) \in dtrans(\varphi)$$

  with $q_j = 0$ for every $j \in [k]$,

(considering the k-tuple of input states with exactly one component is the state 1)

- if there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ then:
  for every $j \in [k]$ with $q_j = 1$:

  $$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

- if there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ then:
  for every $j \in [k]$ with $q_j = 1$:

  $$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

We give an example.

Example 4. Consider the formula $\varphi = (y \in X) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Then:

$$\begin{aligned} dtrans(\varphi) = \{ \; & dt_0((\{\texttt{IDENT}\}, \{y\}, \{X\}), (\,)) = 1 \\ & dt_0((\{\texttt{IDENT}\}, \{y\}, \{\overline{X}\}), (\,)) = 0 \\ & dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = 0 \\ & dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0 \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0 \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1 \\ & dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0,0)) = 0 \\ & dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (1,0)) = 1 \\ & dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0,1)) = 1 \quad \} \end{aligned}$$

□

$\boxed{\varphi = (x == y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

For this formula, we consider for every rank $k \in Rank_\Sigma$ the following:

1. For k-tuple of input states with all components are the state 0, we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x, y\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x, \overline{y}\}$, where the output state of the representation transition for this representation input symbol is 0,

- $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\ \})$, where the output state of the representation transition for this representation input symbol is 0,
- $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0.

These are all the possible representation input symbols which are complete and non-overlapping for the k-tuple of input states where all components are the state 0. Note that we choose to group the input symbols for this k-tuple of input states by deciding for every $\sigma \in \Sigma_k$, whether $sort_{res}(\sigma) \in sort(x)$ or $sort_{res}(\sigma) \notin sort(x)$. Another alternative is to group them by deciding for every $\sigma \in \Sigma_k$, whether $sort_{res}(\sigma) \in sort(y)$ or $sort_{res}(\sigma) \notin sort(y)$, but as in the formula $x \in X$, this also makes no difference, since for this formula $sort(x) = sort(y)$ (cf. definition 23).

2. For k-tuple of input states with exactly one component is the state 1 (and the other components are the state 0), we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}, \overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1.

   These are all the possible representation input symbols which are complete and non-overlapping, with respect to our assumption, for the k-tuple of input states with exactly one component is the state 1. Note that for the first item, it is not necessary to define the representation transition for the representation input symbols with the second components: $\{x, y\}$, $\{x, \overline{y}\}$, and $\{\overline{x}, y\}$, since for this k-tuple of input states it will never be used (recall that the state 1 in this k-tuple of input states means that we have visited a node in a well-marked tree which is labeled by x and y, hence there are no more nodes in that well-marked tree that can be labeled either by x or by y).

Formally, we define $dtrans(\varphi)$ as the smallest set in the following scheme:

For every $k \in Rank_\Sigma$:

(considering the k-tuple of input states with all components are the state 0)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ <u>then</u>:
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x, y\}, \{\}), (q_1, \ldots, q_k)) = 1)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x, \overline{y}\}, \{\}), (q_1, \ldots, q_k)) = 0)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,

  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_k)) = 0)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ <u>then</u>:

$$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\}), (q_1, \ldots, q_k)) = 0) \in dtrans(\varphi)$$

  with $q_j = 0$ for every $j \in [k]$,

(considering the k-tuple of input states with exactly one component is state 1)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ <u>then</u>:
  for every $j \in [k]$ with $q_j = 1$:

$$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}, \overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ <u>then</u>:
  for every $j \in [k]$ with $q_j = 1$:

$$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

We give an example.

Example 5. Consider the formula $\varphi = (y == z) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Then:

$$\begin{aligned} dtrans(\varphi) = \{ \quad & dt_0((\{\texttt{IDENT}\}, \{y, z\}, \{\}), (\,)) = 1, \\ & dt_0((\{\texttt{IDENT}\}, \{y, \overline{z}\}, \{\}), (\,)) = 0, \\ & dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = 0, \\ & dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0, \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0, \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1, \\ & dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 0)) = 0, \\ & dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (1, 0)) = 1, \\ & dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 1)) = 1 \quad \} \end{aligned}$$

□

$\boxed{\varphi = (type_\eta(x)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

For this formula, we consider for every rank $k \in Rank_\Sigma$ the following:

1. For k-tuple of input states with all components are the state 0, we define the representation transitions for the following representation input symbols (if they exist):

   (a) $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) = \eta\}, \{x\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,

   (b) $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) = \eta\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0,

   (c) $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \neq \eta\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0.

   These are all the possible representation input symbols which are complete and non-overlapping for the k-tuple of input states where all components are the state 0.

2. For k-tuple of input states with exactly one component is the state 1 (and the other components are the state 0), we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,

   These are all the possible representation input symbols which are complete and non-overlapping, with respect to our assumption, for the k-tuple of input states with exactly one component is the state 1. Note that for the first item, it is not necessary to define the representation transition for the representation input symbol $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{x\}, \{\})$, since for this k-tuple of input states it will never be used (recall that the state 1 in this k-tuple of input states means that we have visited a node in a well-marked tree with the sort s and which is labeled by x, hence there are no more nodes in that well-marked tree that can be labeled by x).

Formally, we define $dtrans(\varphi)$ as the smallest set by the following scheme:

For every $k \in Rank_\Sigma$:

(considering the k-tuple of input states with all components are the state 0)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) = \eta$, <u>then</u>:
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) = \eta\}, \{x\}, \{\}), (q_1, \ldots, q_k)) = 1)$
    $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,

  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) = \eta\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_k)) = 0)$ $\in dtrans(\varphi)$ with $q_j = 0$ for every $j \in [k]$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \neq \eta$, <u>then</u>: $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \neq \eta\}, \{\}, \{\}), (q_1, \ldots, q_k)) = 0) \in dtrans(\varphi)$ with $q_j = 0$ for every $j \in [k]$,

(considering the k-tuple of input states with exactly one component is state 1)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$, <u>then</u>: for every $j \in [k]$ with $q_j = 1$:

$$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x)\}, \{\overline{x}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$, <u>then</u>: for every $j \in [k]$ with $q_j = 1$:

$$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x)\}, \{\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi),$$

  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

We give an example.

Example 6. Consider the formula $\varphi = type_{\mathtt{IDENT}}(y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Then:

$$\begin{aligned} dtrans(\varphi) \quad = \{ \quad & dt_0((\{\mathtt{IDENT}\}, \{y\}, \{\}), (\,)) = 1, \\ & dt_0((\{\mathtt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = 0, \\ & dt_0((\{\mathtt{INTTYP}, \mathtt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0, \\ & dt_1((\{\mathtt{DSINGLE}\}, \{\}, \{\}), (0)) = 0, \\ & dt_1((\{\mathtt{DSINGLE}\}, \{\}, \{\}), (1)) = 1, \\ & dt_2((\{\mathtt{DECL}, \mathtt{DPAIR}\}, \{\}, \{\}), (0,0)) = 0, \\ & dt_2((\{\mathtt{DECL}, \mathtt{DPAIR}\}, \{\}, \{\}), (1,0)) = 1, \\ & dt_2((\{\mathtt{DECL}, \mathtt{DPAIR}\}, \{\}, \{\}), (0,1)) = 1 \quad \} \end{aligned}$$

□

For these four atomic formulas, we give a comparison of the number of the transitions using our representation with those transitions which are constructed by following the formal construction given in subsection 6.2.2 (where we consider for every input symbol, every state as the input of a transition functions). The comparison is shown in table 7.2.
Note that we have the same number of transitions which are constructed following the formal construction given in subsection 6.2.2 (in the column "Non-represented"). Those transitions are obtained from the following:

MSO* Formula	Non-represented	Represented
$label_\sigma(x)$	26	10
$x \in X$	26	9
$x == y$	26	9
$type_\eta(x)$	26	8

Table 7.2: A comparison of the number of transitions (represented and non-represented) for the formulas $label_\sigma(x)$, $x \in X$, $x == y$, and $type_\eta(x)$

- 8 transitions having the input symbols with the first component: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, where we consider 2 node variables ($y^{\{\texttt{Ident}\}}$ and $z^{\{\texttt{Ident}\}}$), 1 node set variable ($X^{\{\texttt{Ident}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}$, where we consider no node variables, no node set variable, and two combinations of input states, i.e. (0) and (1)
- 4 transitions having the input symbols with the first component: $\texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}$, where we consider no node variable, no node set variable, and four combinations of input states, i.e. (0,0), (0,1), (1,0), and (1,1),
- 8 transitions having the input symbols with the first component: $\texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}$, where we consider 1 node variable ($x^{\{\texttt{Decl}\}}$), no node set variable, and four combinations of input states, i.e. (0,0), (0,1), (1,0), and (1,1).

Now we consider the following two atomic formulas which we handle similarly:

- $\varphi = edge_i(x, y)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$
- $\varphi = over(x, y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$

We define, for every rank $k \in Rank_\Sigma$, the representation transitions for these two formulas with the following input states:

- all components of their k-tuples of input states are the state 0,
- exactly one component of their k-tuples of input states is the state 1, whereas the other components are the states 0. Recall from subsection 6.2.2 that the state 1 is reached, if we have visited a node labeled by y. And since we consider well-marked trees on which the automata run, then

there exists exactly one node which is labeled by y. This means we do not need to represent the transitions with more than one component of their k-tuples of input states are the state 1, since those transitions will never be used due to our assumption.

Note that for the formula $edge_i(x, y)$, following its operational semantics defined in subsection 6.2.2, we shall distinguish in the definition of the representation transitions for this k-tuple of input states into the cases whether the state 1 is at the i-th position (for $i \leq k$) or the state 1 is not at the i-th position. On the other hand, these two cases have no difference for the formula $over(x, y)$. Hence, for the formula $over(x, y)$ we do not make such distinction.

- exactly one component of their k-tuples of input states is the state 2, whereas the other components are the states 0. We also do not need to represent the transitions with more than one component of their k-tuples of input states are the state 2, since state 2 is reached, if we have visited a node labeled by x (after visited a node labeled by y beforehand) and in well-marked trees on which the automata run there exists exactly one node which is labeled by x.

These are all the possible input states of the representation transitions we may have for both atomic formulas with respect to our assumption. By considering these possible input states, in the following we discuss how we define the representation of the transitions for these two formulas. In particular, we show how we group the input symbols from the extended ranked alphabet (considering the behavior of the transitions) by means of our representation input symbols, such that the transitions are defined completely and non-overlappingly. The only difference in grouping input symbols between these two formulas is only when we consider the k-tuple of input states which has exactly one component of the state 1.

$\boxed{\varphi = (edge_i(x, y)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

For this formula, we consider for every rank $k \in Rank_\Sigma$ the following:

1. For k-tuple of input states with all components are the state 0, we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y)\}, \{y\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y)\}, \{\overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(y)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0.

   These are all the possible representation input symbols which are complete and non-overlapping for the k-tuple of input states where all components are the state 0.

2. For k-tuple of input states with exactly one component is the state 1 at the i-th position (for $i \leq k$), we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \wedge sort_{res}(\sigma) \in sort(y) \wedge sort_{in_i}(\sigma) \in sort(y)\}, \{x, \overline{y}\}, \{\})$,
     where the output state of the representation transition for this representation input symbol is 2,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \wedge sort_{res}(\sigma) \in sort(y) \wedge sort_{in_i}(\sigma) \in sort(y)\}, \{\overline{x}, \overline{y}\}, \{\})$,
     where the output state of the representation transition for this representation input symbol is 0,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \wedge sort_{res}(\sigma) \notin sort(y) \wedge sort_{in_i}(\sigma) \in sort(y)\}, \{x\}, \{\})$,
     where the output state of the representation transition for this representation input symbol is 2,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \wedge sort_{res}(\sigma) \notin sort(y) \wedge sort_{in_i}(\sigma) \in sort(y)\}, \{\overline{x}\}, \{\})$,
     where the output state of the representation transition for this representation input symbol is 0,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \wedge sort_{res}(\sigma) \in sort(y) \wedge sort_{in_i}(\sigma) \in sort(y)\}, \{\overline{y}\}, \{\})$,
     where the output state of the representation transition for this representation input symbol is 0,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \wedge sort_{res}(\sigma) \notin sort(y) \wedge sort_{in_i}(\sigma) \in sort(y)\}, \{\}, \{\})$,
     where the output state of the representation transition for this representation input symbol is 0.

   Note that for this k-tuple of input states, we check whether we are visiting a node which is labeled by x, after having visited its direct i-th successor node labeled by y. The grouping of input symbols is made concerning the sorts of the variables x and y. Moreover, for this k-tuple of input states, we only represent the transitions where the i-th input sort of an input symbol (which is the sort of the i-th successor node of this input symbol) is in the sort of y. The reason is that, the state 1 at the i-th position is reached only if the direct i-th successor node of the input symbol we consider is labeled by y. Then, if the sort of the direct i-th successor node of the input symbol we consider is not in the sort of y, this i-th successor node can not be labeled by y. Hence, it is also not the case that the input state at the i-th position of the k-tuple is the state 1.

   By this way of grouping, the representation transitions for this k-tuple of input states are defined completely and non-overlappingly, with respect to our assumption.

3. For k-tuple of input states with exactly one component is the state 1 at the m-th position, where $m \in [k]$ and $m \neq i$, we define the representation transitions for the following representation input symbols (if they exist):

- $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y) \ \wedge \ sort_{in_m}(\sigma) \in sort(y)\}, \{\overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0,
- $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(y) \ \wedge \ sort_{in_m}(\sigma) \in sort(y)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0.

Note that for this k-tuple of input states, in grouping the input symbols we do not distinguish whether they are labeled by x or not, since this k-tuple of input states means that we have visited a successor node which is labeled by y and this node is not the i-th successor node. Hence, the fact that we are visiting a node which is labeled or not labeled by x afterward is not significant (since both give the same output state). The grouping of input symbols is made concerning the sorts of the variable y. Moreover, for the same reason as in the case where the state 1 is at the i-th position, we only represent the transitions where the m-th input sort of an input symbol (which is the sort of the m-th successor node of this input symbol) is in the sort of y.

By this way of grouping, the representation transitions for this k-tuple of input states are defined completely and non-overlappingly, with respect to our assumption.

4. For k-tuple of input states with exactly one component is state 2 (and the other components are the state 0), we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \ \wedge \ sort_{res}(\sigma) \in sort(y)\}, \{\overline{x}, \overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \ \wedge \ sort_{res}(\sigma) \notin sort(y)\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \ \wedge \ sort_{res}(\sigma) \in sort(y)\}, \{\overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \ \wedge \ sort_{res}(\sigma) \notin sort(y)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,

   The grouping of input symbols is made concerning the sorts of the variables x and y. By this way of grouping, the representation transitions for this k-tuple of input states are defined completely and non-overlappingly, with respect to our assumption. No representation transitions with representation input symbols labeled by variable *without* overline (either x or y) are needed, since by our assumption those transitions are never used.

Formally, we define $dtrans(\varphi)$ as the smallest set by the following scheme:

For every $k \in Rank_\Sigma$:

(considering the k-tuple of input states with all components are the state 0)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(y)$ <u>then</u>:
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y)\}, \{y\}, \{\}), (q_1, \ldots, q_k)) = 1)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,
  - $(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y)\}, \{\overline{y}\}, \{\}), (q_1, \ldots, q_k)) = 0)$ $\in dtrans(\varphi)$
    with $q_j = 0$ for every $j \in [k]$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:

  $$(dt_k((\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(y)\}, \{\}, \{\}), (q_1, \ldots, q_k)) = 0) \in dtrans(\varphi)$$

  with $q_j = 0$ for every $j \in [k]$,

(considering the k-tuple of input states with exactly one component is the state 1 at the i-th position, for $i \leq k$)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$, $sort_{res}(\sigma) \in sort(y)$, and $sort_{in_i}(\sigma) \in sort(y)$ <u>then</u>:
  Let $ielem_{xy} = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \ \wedge \ sort_{res}(\sigma) \in sort(y) \ \wedge \ sort_{in_i}(\sigma) \in sort(y)\}$.
  - $(dt_k((ielem_{xy}, \{x, \overline{y}\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 2) \in dtrans(\varphi)$
    with $q_i = 1$ and for every $m \in [k] \setminus \{i\}$: $q_m = 0$,
  - $(dt_k((ielem_{xy}, \{\overline{x}, \overline{y}\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 0) \in dtrans(\varphi)$
    with $q_i = 1$ and for every $m \in [k] \setminus \{i\}$: $q_m = 0$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$, $sort_{res}(\sigma) \notin sort(y)$, and $sort_{in_i}(\sigma) \in sort(y)$ <u>then</u>:
  Let $ielem_x = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \ \wedge \ sort_{res}(\sigma) \notin sort(y) \ \wedge \ sort_{in_i}(\sigma) \in sort(y)\}$.
  - $(dt_k((ielem_x, \{x\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 2) \in dtrans(\varphi)$
    with $q_i = 1$ and for every $m \in [k] \setminus \{i\}$: $q_m = 0$,
  - $(dt_k((ielem_x, \{\overline{x}\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 0) \in dtrans(\varphi)$
    with $q_i = 1$ and for every $m \in [k] \setminus \{i\}$: $q_m = 0$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$, $sort_{res}(\sigma) \in sort(y)$, and $sort_{in_i}(\sigma) \in sort(y)$ <u>then</u>:
  Let $ielem_y = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \ \wedge \ sort_{res}(\sigma) \in sort(y) \ \wedge \ sort_{in_i}(\sigma) \in sort(y)\}$.
  $(dt_k((ielem_y, \{\overline{y}\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 0) \in dtrans(\varphi)$
  with $q_i = 1$ and for every $m \in [k] \setminus \{i\}$: $q_m = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$, $sort_{res}(\sigma) \notin sort(y)$, and $sort_{in_i}(\sigma) \in sort(y)$ <u>then</u>:
  Let $ielem = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \ \wedge \ sort_{res}(\sigma) \notin sort(y) \ \wedge \ sort_{in_i}(\sigma) \in sort(y)\}$.
  $(dt_k((ielem, \{\}, \{\}), (q_1, \ldots, q_i, \ldots, q_k)) = 0) \in dtrans(\varphi)$
  with $q_i = 1$ and for every $m \in [k] \setminus \{i\}$: $q_m = 0$,

(considering the k-tuple of input states with exactly one component is the state 1 at the m-th position, where $m \in [k]$ and $m \neq i$)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(y)$ <u>then</u>:
  For every $m \in [k] \setminus \{i\}$ with $sort_{in_m}(\sigma) \in sort(y)$ and $q_m = 1$:
  Let $melem_y = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y) \ \wedge \ sort_{in_m}(\sigma) \in sort(y)\}$.
  $(dt_k((melem_y, \{\overline{y}\}, \{\}), (q_1, \ldots, q_m, \ldots, q_k)) = 0) \in dtrans(\varphi)$,
  where for every $n \in [k] \setminus \{m\}$: $q_n = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:
  For every $m \in [k] \setminus \{i\}$ with $sort_{in_m}(\sigma) \in sort(y)$ and $q_m = 1$:
  Let $melem = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(y) \ \wedge \ sort_{in_m}(\sigma) \in sort(y)\}$.
  $(dt_k((melem, \{\}, \{\}), (q_1, \ldots, q_m, \ldots, q_k)) = 0) \in dtrans(\varphi)$,
  where for every $n \in [k] \setminus \{m\}$: $q_n = 0$,

(considering the k-tuple of input states with exactly one component is the state 2)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ and $sort_{res}(\sigma) \in sort(y)$ <u>then</u>:
  Let $jelem_{xy} = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \ \wedge \ sort_{res}(\sigma) \in sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((jelem_{xy}, \{\overline{x}, \overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $j' \in [k] \setminus \{j\}$: $q_{j'} = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ and $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:
  Let $jelem_x = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \ \wedge \ sort_{res}(\sigma) \notin sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((jelem_x, \{\overline{x}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $j' \in [k] \setminus \{j\}$: $q_{j'} = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ and $sort_{res}(\sigma) \in sort(y)$ <u>then</u>:
  Let $jelem_y = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \ \wedge \ sort_{res}(\sigma) \in sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((jelem_y, \{\overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $j' \in [k] \setminus \{j\}$: $q_{j'} = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ and $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:
  Let $jelem = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \ \wedge \ sort_{res}(\sigma) \notin sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((jelem, \{\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $j' \in [k] \setminus \{j\}$: $q_{j'} = 0$,

We give an example.

Example 7. Consider the formula $\varphi = edge_1(x, y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Then:

$$\begin{aligned} dtrans(\varphi) \quad = \{ \quad & dt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) = 1, \\ & dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = 0, \\ & dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0, \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0, \\ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (2)) = 2, \\ & dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 0)) = 0, \\ & dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (1, 0)) = 2, \\ & dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (1, 0)) = 0, \\ & dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (2, 0)) = 2, \\ & dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0, 2)) = 2, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (2, 0)) = 2, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0, 2)) = 2 \quad \} \end{aligned}$$

Note that we do not have the following representation transitions:

- $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1$, since the sort of the first successor node of `DSINGLE` is `Decl`, and such node can not be labeled by y, which has the sort $\{\texttt{Ident}\}$.
- $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (0, 1)) = 0$, since the sort of the second successor node of `DECL` is `TypExp`, and such node can not be labeled by y, which has the sort $\{\texttt{Ident}\}$.
- $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (1, 0)) = 1$, since the sort of the first successor node of `DPAIR` is `Decl`, and such node can not be labeled by y, which has the sort $\{\texttt{Ident}\}$.
- $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0, 1)) = 0$, since the sort of the second successor node of `DPAIR` is `DList`, and such node can not be labeled by y, which has the sort $\{\texttt{Ident}\}$. □

$\boxed{\varphi = (x\ over\ y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$

For this formula, we consider for every rank $k \in Rank_\Sigma$ the following:

1. For k-tuple of input states with all components are the state 0, we define the representation transitions for the following representation input symbols (if they exist):
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y)\}, \{y\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(y)\}, \{\overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0,

- $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 0.

These are all the possible representation input symbols which are complete and non-overlapping for the k-tuple of input states where all components are the state 0.

2. For k-tuple of input states with exactly one component is the state 1, we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}, \{x, \overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,
   - $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}, \{\overline{x}, \overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)\}, \{x\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,
   - $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}, \{\overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,
   - $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 1,

   Note that for this k-tuple of input states, we check whether we are visiting a node which is labeled by x, after having visited a node labeled by y. The grouping of input symbols is made concerning the sorts of the variables x and y. In contrast to the formula $\mathit{edge}_i(x, y)$, here we do not need to consider a particular node successor, since the state 1 in the input state is reached whenever we have visited a node (not necessarily a particular direct successor) labeled by y.

   By this way of grouping, the representation transitions for this k-tuple of input states are defined completely and non-overlappingly, with respect to our assumption.

3. For k-tuple of input states with exactly one component is state 2 (and the other components are the state 0), we define the representation transitions for the following representation input symbols (if they exist):

   - $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}, \{\overline{x}, \overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,

- $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)\}, \{\overline{x}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,
- $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}, \{\overline{y}\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,
- $(\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)\}, \{\}, \{\})$, where the output state of the representation transition for this representation input symbol is 2,

The grouping of input symbols is made concerning the sorts of the variables x and y. By this way of grouping, the representation transitions for this k-tuple of input states are defined completely and non-overlappingly, with respect to our assumption. No representation transitions with representation input symbols labeled by variable *without* overline (either x or y) are needed, since by our assumption those transitions are never used.

Formally, we define $\mathit{dtrans}(\varphi)$ as the smallest set by the following scheme:

For every $k \in \mathit{Rank}_\Sigma$:

(considering the k-tuple of input states with all components are the state 0)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $\mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)$ <u>then</u>:
  - $(\mathit{dt}_k((\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}, \{y\}, \{\}), (q_1, \ldots, q_k)) = 1)$ $\in \mathit{dtrans}(\varphi)$ with $q_j = 0$ for every $j \in [k]$,
  - $(\mathit{dt}_k((\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}, \{\overline{y}\}, \{\}), (q_1, \ldots, q_k)) = 0)$ $\in \mathit{dtrans}(\varphi)$ with $q_j = 0$ for every $j \in [k]$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $\mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)$ <u>then</u>:

  $$(\mathit{dt}_k((\{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \notin \mathit{sort}(y)\}, \{\}, \{\}), (q_1, \ldots, q_k)) = 0) \in \mathit{dtrans}(\varphi)$$

  with $q_j = 0$ for every $j \in [k]$,

(considering the k-tuple of input states with exactly one component is the state 1)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $\mathit{sort}_{res}(\sigma) \in \mathit{sort}(x)$ and $\mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)$ <u>then</u>: Let $\mathit{elem1}_{xy} = \{\sigma \in \Sigma_k \mid \mathit{sort}_{res}(\sigma) \in \mathit{sort}(x) \ \wedge \ \mathit{sort}_{res}(\sigma) \in \mathit{sort}(y)\}$.
  - for every $j \in [k]$ with $q_j = 1$: $(\mathit{dt}_k((\mathit{elem1}_{xy}, \{x, \overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in \mathit{dtrans}(\varphi)$, where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,
  - for every $j \in [k]$ with $q_j = 1$: $(\mathit{dt}_k((\mathit{elem1}_{xy}\}, \{\overline{x}, \overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in \mathit{dtrans}(\varphi)$, where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ and $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:
  Let $elem1_x = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \;\wedge\; sort_{res}(\sigma) \notin sort(y)\}$.
  - for every $j \in [k]$ with $q_j = 1$:
    $(dt_k((elem1_x, \{x\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
    where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,
  - for every $j \in [k]$ with $q_j = 1$:
    $(dt_k((elem1_x, \{\overline{x}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi)$,
    where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ and $sort_{res}(\sigma) \in sort(y)$ <u>then</u>:
  Let $elem1_y = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \;\wedge\; sort_{res}(\sigma) \in sort(y)\}$.
  For every $j \in [k]$ with $q_j = 1$:
  $(dt_k((elem1_y, \{\overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi)$,
  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ and $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:
  Let $elem1 = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \;\wedge\; sort_{res}(\sigma) \notin sort(y)\}$.
  For every $j \in [k]$ with $q_j = 1$:
  $(dt_k((elem1, \{\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 1) \in dtrans(\varphi)$,
  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

(considering the k-tuple of input states with exactly one component is the state 2)

- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ and $sort_{res}(\sigma) \in sort(y)$ <u>then</u>:
  Let $elem2_{xy} = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \;\wedge\; sort_{res}(\sigma) \in sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((elem2_{xy}, \{\overline{x}, \overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \in sort(x)$ and $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:
  Let $elem2_x = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \in sort(x) \;\wedge\; sort_{res}(\sigma) \notin sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((elem2_x, \{\overline{x}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ and $sort_{res}(\sigma) \in sort(y)$ <u>then</u>:
  Let $elem2_y = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \;\wedge\; sort_{res}(\sigma) \in sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((elem2_y, \{\overline{y}\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,
- <u>if</u> there is $\sigma \in \Sigma_k$ with $sort_{res}(\sigma) \notin sort(x)$ and $sort_{res}(\sigma) \notin sort(y)$ <u>then</u>:
  Let $elem2 = \{\sigma \in \Sigma_k \mid sort_{res}(\sigma) \notin sort(x) \;\wedge\; sort_{res}(\sigma) \notin sort(y)\}$.
  For every $j \in [k]$ with $q_j = 2$:
  $(dt_k((elem2, \{\}, \{\}), (q_1, \ldots, q_j, \ldots, q_k)) = 2) \in dtrans(\varphi)$,
  where for every $i \in [k] \setminus \{j\}$: $q_i = 0$,

We give an example.

Example 8. Consider the formula $\varphi = (x\ over\ y) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$. Then:

$$
\begin{aligned}
dtrans(\varphi) \quad = \{ \quad & dt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) = 1,\\
& dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = 0,\\
& dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0,\\
& dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0,\\
& dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1,\\
& dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (2)) = 2,\\
& dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0,0)) = 0,\\
& dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (1,0)) = 2,\\
& dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (0,1)) = 2,\\
& dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (1,0)) = 1,\\
& dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0,1)) = 1,\\
& dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (2,0)) = 2,\\
& dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0,2)) = 2,\\
& dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (1,0)) = 1,\\
& dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,1)) = 1,\\
& dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (2,0)) = 2,\\
& dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,2)) = 2 \quad \}
\end{aligned}
$$

□

We give a comparison of the transitions for these two formulas, by comparing the number of transitions using our representation and those which are constructed by following the formal construction given in subsection 6.2.2, where we consider for every input symbol, every state as the input of a transition function. This is shown in table 7.3.

MSO* Formula	Non-represented	Represented
$edge_i(x, y)$	42	12
$over(x, y)$	42	17

Table 7.3: A comparison of the number of transitions (represented and non-represented) for the formulas $edge_i(x, y)$ and $over(x, y)$

Note that we have the same number of transitions which are constructed following the formal construction given in subsection 6.2.2 (in the column "Non-represented"). Those transitions are obtained from the following:

- 8 transitions having the input symbols with the first component: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, where we consider 2 node variables ($y^{\{\texttt{Ident}\}}$ and $z^{\{\texttt{Ident}\}}$), 1 node set variable ($X^{\{\texttt{Ident}\}}$), and no input state,

- 2 transitions having the input symbols with the first component: `INTTYP`$^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: `BOOLTYP`$^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 3 transitions having the input symbols with the first component: `DSINGLE`$^{(\texttt{Decl},\texttt{DList})}$, where we consider no node variables, no node set variable, and three combinations of input states, i.e. (0), (1), and (2),
- 9 transitions having the input symbols with the first component: `DPAIR`$^{(\texttt{Decl DList},\texttt{DList})}$, where we consider no node variable, no node set variable, and nine combinations of input states, i.e. (0,0), (0,1), (1,0), (1,1), (0,2), (2,0), (1,2), (2,1), and (2,2),
- 18 transitions having the input symbols with the first component: `DECL`$^{(\texttt{Ident TypExp},\texttt{Decl})}$, where we consider 1 node variable ($x^{\{\texttt{Decl}\}}$), no node set variable, and nine combinations of input states, i.e. (0,0), (0,1), (1,0), (1,1), (0,2), (2,0), (1,2), (2,1), and (2,2).

For all atomic formulas we have discussed so far, we define the representation transitions in the following steps:

1. We define all possible configurations of input states with respect to our assumption.
2. For every possible configuration of input states, we group input symbols by means of our representation of input symbols. The idea of grouping this input symbols is to group the transitions which have the same behavior, i.e. give the same output state on the same configuration of input states. The input symbols are grouped, such that the transitions for a given configuration of input states are represented completely and non-overlappingly with respect to our assumption.

In the following, we consider the representation transitions of an attribute formula. Compared to the other atomic formulas, we define the representation transitions of an attribute formula slightly different.

$$\boxed{(\varphi \in T^{Bool}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}) \in MSO^*(\Sigma,\mathcal{V}_1,\mathcal{V}_2)_{\mathcal{A},\mathcal{F}}}$$

Before we define the representation transitions of this formula, we give some observations which distinguish the way we represent the transitions for this formula from the other atomic formulas:

- Observation 1.
  In the atomic formulas we have discussed so far, it often happens that the presence of some node variables (which occur in the formula) as a part of input symbol in a transition have no influence to the output state. As an example, for the atomic formula $label_\sigma(x)$, the output state of the

transition with the input symbol not containing σ does not depend on the occurrence of x in that input symbol.

This is not the case for an attribute formula. The presence of a node variable (that occurs in the formula) as a part of input symbol in a transition is significant to determine the output state. The reasons are the following:

- If a node variable occurs as a label of a node, then this is the first and the only occurrence of this node variable in a well-marked tree. Or, in the other words, we never visited another node before with such node variable as its label. This is reflected by a dummy value for the value of this node variable in the input state. The occurrence of this node variable as a node's label changes the value of the node variable from a dummy value into ε of the Dewey notation, which is reflected in the output state (recall the transition function of the automaton for an attribute formula defined in subsection 6.2.2).
- On the other hand, if a node variable does not occur as a label of a node and such node variable has never been a label of the nodes visited before, then the value of this node variable remains as a dummy value, which is reflected in the output state.

Since the presence of a node variable (that occurs in an attribute formula) as a part of input symbol in a transition determines the output state, then for every node variable that occur in an attribute formula, the fact whether a node variable is a part of the input symbol of a transition should be clearly represented in the representation input symbol of a representation transition. This is realized by having combinations of representation node variables with overlines or without overlines in the representation input symbol. For example, if there are two node variables occur in an attribute formula, e.g. x and y, then there are four combinations for representing the occurrences of x and y as a part of the input symbol:

- (x,y), which means that both x and y are labels in the input symbol,
- $(x,\overline{y})$, which means that only x (and not y) is a label in the input symbol,
- $(\overline{x},y)$, which means that only y (and not x) is a label in the input symbol,
- $(\overline{x},\overline{y})$, which means that both x and y are not labels in the input symbol.

• <u>Observation 2</u>.
Having the first observation, given a representation input symbol with the knowledge whether a node variable is a part or not a part of this representation input symbol, we can derive all possible configurations of input states for the representation transitions of this representation input symbol with respect to our assumption. Suppose we are considering a k-tuple of input states, where k is the rank of the input symbol, then the value of a node variable in this k-tuple of input states is derived by the following fact:

- If a node variable, say x, is known to be a part of an input symbol, then as we have mentioned in the first observation, this occurrence of x will be the first and the only occurrence in a well-marked tree. This means the value of x, i.e. the position of the node labeled by x, in every input state must be a dummy value (which means that we have never seen x as a label of the nodes we have visited so far).
- On the other hand, if a node variable, say x, is not a part of an input symbol, then there are two possibilities of value for x in every component of this k-tuple of input states:
  - a dummy value, which means that a node labeled by x has never been visited before,
  - a Dewey notation representing the path of a node labeled by x we have already visited. Since we consider a well-marked tree, then there is exactly one such node and consequently there is exactly one input state with the value of the node variable x is a Dewey notation (and the value of node variable x in the other input states are dummy values).

Based on these two observations, we shall define the representation transitions for an attribute formula. As a running example, consider the attribute formula:

$$\varphi = gt(\langle often, x\rangle(\langle name, y\rangle), null) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

We define the representation transitions for the attribute formula φ in the following steps:

1. We collect all different node variables that occur in φ. For our running example, we have the set $\{x, y\}$.
2. **Defining the representation input symbols**.
   From this set of node variables, for every $k \in Rank_\Sigma$ we group every symbol of Σ_k to derive a representation input symbol, such that for every symbol of Σ_k, it is a part of exactly one representation input symbol.

   Since every representation input symbol should respect the definition 28 (in particular regarding the result sort of the symbol from Σ and the sort of representation node variables), we propose the following idea for grouping such symbols:

   - We collect all complete two partitions (CP) of the set of node variables that occur in φ. For our running example, we have:
     - CP_1 consists of the first partition: $\{\}$ and the second partition: $\{x, y\}$,
     - CP_2 consists of the first partition: $\{x\}$ and the second partition: $\{y\}$,
     - CP_3 consists of the first partition: $\{y\}$ and the second partition: $\{x\}$,
     - CP_4 consists of the first partition: $\{x, y\}$, the second partition: $\{\}$.

- For every complete two partitions, we group for every $k \in \mathit{Rank}_\Sigma$, every symbol of Σ_k such that its result sort is in the sort of every node variable of the first partition, but is *not* in the sort of any variable of the second partition (if such symbols exist). In this way of grouping, each symbol of Σ_k can only be a member of exactly one group. The groups of symbols of Σ_k for every $k \in \mathit{Rank}_\Sigma$ are shown in table 7.4, table 7.5, and table 7.6.

CP	1^{st} Partition	2^{nd} Partition	Group of Σ_0-symbols
CP_1	$\{\}$	$\{x, y\}$	$\{\texttt{INTTYP}, \texttt{BOOLTYP}\}$
CP_2	$\{x\}$	$\{y\}$	$\{\}$
CP_3	$\{y\}$	$\{x\}$	$\{\texttt{IDENT}\}$
CP_4	$\{x, y\}$	$\{\}$	$\{\}$

Table 7.4: The groups of symbols of $\Sigma_0 = \{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}$

CP	1^{st} Partition	2^{nd} Partition	Group of Σ_1-symbols
CP_1	$\{\}$	$\{x, y\}$	$\{\texttt{DSINGLE}\}$
CP_2	$\{x\}$	$\{y\}$	$\{\}$
CP_3	$\{y\}$	$\{x\}$	$\{\}$
CP_4	$\{x, y\}$	$\{\}$	$\{\}$

Table 7.5: The groups of symbols of $\Sigma_1 = \{\texttt{DSINGLE}\}$

CP	1^{st} Partition	2^{nd} Partition	Group of Σ_2-symbols
CP_1	$\{\}$	$\{x, y\}$	$\{\texttt{DPAIR}\}$
CP_2	$\{x\}$	$\{y\}$	$\{\texttt{DECL}\}$
CP_3	$\{y\}$	$\{x\}$	$\{\}$
CP_4	$\{x, y\}$	$\{\}$	$\{\}$

Table 7.6: The groups of symbols of $\Sigma_2 = \{\texttt{DPAIR}, \texttt{DECL}\}$

- A representation input symbol for every $k \in \mathit{Rank}_\Sigma$ can now be constructed by using the non-empty group of symbols from Σ_k together with the first partition of the corresponding CP from which the symbols of Σ_k are grouped. Note that for every representation input symbol, we can always have $\{\}$ as the set of representation node set variables. This is because the output state of a transition for an attribute formula does not depend any occurrence of node set variable in an input symbol (recall from our first observation, that only the presence of a node variable in an input symbol have influence to the output state). Indeed, no node set variable is contained in an attribute formula. For our running example, we have the following:
  - for the rank 0:
    $(\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\})$ and $(\{\texttt{IDENT}\}, \{y\}, \{\})$,
  - for the rank 1:
    $(\{\texttt{DSINGLE}\}, \{\}, \{\})$,

– for the rank 2:
$(\{\texttt{DPAIR}\}, \{\}, \{\})$ and $(\{\texttt{DECL}\}, \{x\}, \{\})$,

These are not all of the representation input symbols we have, since if we reconsider our first observation, we know that the presence of a node variable in an input symbol of a transition determines the output state. Hence, this should also be reflected in our representation input symbols. This is done by constructing for every representation input symbol in which the representation node variable occur without overline, another representation input symbols with variants in their representation node variables (by having or not having overlines). For example, for the representation input symbol $(\{\texttt{IDENT}\}, \{y\}, \{\})$ in our running example (which means that y occurs in the input symbol together with `IDENT`), we construct another representation input symbol, i.e. $(\{\texttt{IDENT}\}, \{\overline{y}\}, \{\})$ (which means that y does not occur in the input symbol together with `IDENT`). All representation input symbols for our running example is shown in table 7.7.

Rank	Representation Input Symbol
0	$(\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\})$
	$(\{\texttt{IDENT}\}, \{y\}, \{\})$
	$(\{\texttt{IDENT}\}, \{\overline{y}\}, \{\})$
1	$(\{\texttt{DSINGLE}\}, \{\}, \{\})$
2	$(\{\texttt{DPAIR}\}, \{\}, \{\})$
	$(\{\texttt{DECL}\}, \{x\}, \{\})$
	$(\{\texttt{DECL}\}, \{\overline{x}\}, \{\})$

Table 7.7: Representation input symbols for $gt(\langle often, x\rangle(\langle name, y\rangle), null)$

3. **Defining the input states**.
Having all representation input symbols, by our second observation, we can derive for every representation input symbol all possible configurations of input states. Recall from section 7.2, that we have an l-tuple as a state, where l is the number of different node variables in φ. Moreover, the components of this l-tuple is represented symbolically. In an input state, an l-tuple can have a component which is either a dummy value (represented as d_x, where x is a node variable in an attribute formula) or a general representation of Dewey notation (which is nd_x, where x is a node variable in an attribute formula). Here we need a general representation nd_x (and not the other two particular representations) as a component of an l-tuple, since as an input state any Dewey notation can be a component of an l-tuple. And nd_x as a general representation of Dewey notation represents all of them (cf. section 7.2).

   Since every element of an l-tuple corresponds to the value of a node variable in an attribute formula, we fix the order of elements in the l-tuple by following the order of node variables appearing in the formula from left to right.

We define now how we can derive all configurations of input states for every representation input symbol based on our second observation:

- If a representation input symbol contains a representation node variable without overline, say x, then the value of the j-th component (where j is the position in l-tuple, assigned for the value of the node variable x) of every input state is a representation of a dummy value, d_x. This holds for every representation node variable appears in the representation input symbol.
- Otherwise, there is *at most* one input state with the value of the j-th component (where j is the position in l-tuple, assigned for the value of the node variable x) is a general representation of Dewey notation, nd_x.

In table 7.8 we give for every representation input symbol of our running example, all possible configurations of input states.

Rank	Representation Input Symbol	Input States Configuration
0	$(\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\})$	$(\,)$
	$(\{\texttt{IDENT}\}, \{y\}, \{\})$	$(\,)$
	$(\{\texttt{IDENT}\}, \{\overline{y}\}, \{\})$	$(\,)$
1	$(\{\texttt{DSINGLE}\}, \{\}, \{\})$	$((d_x, d_y))$ $((d_x, nd_y))$ $((nd_x, d_y))$ $((nd_x, nd_y))$
2	$(\{\texttt{DPAIR}\}, \{\}, \{\})$	$((d_x, d_y), (d_x, d_y))$ $((d_x, d_y), (d_x, nd_y))$ $((d_x, d_y), (nd_x, d_y))$ $((d_x, d_y), (nd_x, nd_y))$ $((d_x, nd_y), (d_x, d_y))$ $((d_x, nd_y), (nd_x, d_y))$ $((nd_x, d_y), (d_x, d_y))$ $((nd_x, d_y), (d_x, nd_y))$ $((nd_x, nd_y), (d_x, d_y))$
	$(\{\texttt{DECL}\}, \{x\}, \{\})$	$((d_x, d_y), (d_x, d_y))$ $((d_x, d_y), (d_x, nd_y))$ $((d_x, nd_y), (d_x, d_y))$
	$(\{\texttt{DECL}\}, \{\overline{x}\}, \{\})$	$((d_x, d_y), (d_x, d_y))$ $((d_x, d_y), (d_x, nd_y))$ $((d_x, d_y), (nd_x, d_y))$ $((d_x, d_y), (nd_x, nd_y))$ $((d_x, nd_y), (d_x, d_y))$ $((d_x, nd_y), (nd_x, d_y))$ $((nd_x, d_y), (d_x, d_y))$ $((nd_x, d_y), (d_x, nd_y))$ $((nd_x, nd_y), (d_x, d_y))$

Table 7.8: Input states configurations for $gt(\langle often, x\rangle(\langle name, y\rangle), null)$

4. **Defining the output state**.
   The output state of a transition is determined as the following:
   - if we have a representation node variable x_j in the representation input symbol (this is the case where we are visiting a node labeled by x_j now), then ε_{x_j} is the value of node variable x_j in the l-tuple output state,
   - if there is exactly one input state, e.g. the i-th input state, with the value of node variable x_j is nd_{x_j} (this is the case where we have visited a node labeled by x_j), then $i.nd_{x_j}$ is the value of node variable x_j in the l-tuple of output state. This describes that the node which is labeled by x_j can be found by following the i-th successor and the path information represented in nd_{x_j}.
   - otherwise (this is the case where we have not been visiting a node labeled by x_j), then d_{x_j} is the value of node variable x_j in the l-tuple of output state.

By defining the representation input symbol and all possible configuration of input states in this way, we obtain a complete and non-overlapping representation transitions with respect to our assumption.

In the following, we formally define what we have just described.

Let $Var(\varphi)$ be the set of all node variables in φ and $l = card(Var(\varphi))$.

We define CP_φ as a set of all complete two partitions of $Var(\varphi)$, where each complete two partition is represented as a pair:

$$CP_\varphi = \{(P_1, P_2) \mid P_1 \subseteq Var(\varphi) \ \wedge \ P_2 = Var(\varphi) \setminus P_1\}.$$

Let $k \in Rank_\Sigma$. We define $Grp_{\varphi,k} : CP_\varphi \to \mathcal{P}(\Sigma_k)$ as a function that given a complete two partition $(P_1, P_2) \in CP_\varphi$, it groups the symbols from Σ_k such that their result sorts are in the sort of every node variables of P_1, but *not* in the sort of any variable of P_2:

$$\begin{aligned} Grp_{\varphi,k}((P_1,P_2)) \quad = \quad & \{\sigma \in \Sigma_k \mid \forall x \in P_1.\ sort_{res}(\sigma) \in sort(x) \ \wedge \\ & \forall x \in P_2.\ sort_{res}(\sigma) \notin sort(x)\}. \end{aligned}$$

Let $S \subseteq \mathcal{V}_1$, $\overline{S} = \{\overline{x}^{\widetilde{\eta}} \mid x^{\widetilde{\eta}} \in S\}$, and $\widehat{S} = S \cup \overline{S}$.

We define a function, that given a set of node variables R, it computes all sets R' of representation node variables where every set R' describes the presence of every node variable in R (which is represented by having or not having overline), $Pr : \mathcal{P}(Var(\varphi)) \to \mathcal{P}(\mathcal{P}(\widehat{Var(\varphi)}))$ such that:

$$Pr(R) = \{R' \subset \widehat{R} \mid card(R') = card(R) \ \wedge \ \forall x \in R.\ (x \in R' \ \vee \ \overline{x} \in R')\}.$$

To compute all configurations of input states (k-tuple of input states, where $k \in Rank_\Sigma$), we define the function $InSt_{\varphi,k} : \mathcal{P}(\widehat{Var(\varphi)}) \to \mathcal{P}(St(\varphi)^k)$, that given a set of representation node variables E_1, it computes all possible k-tuples of input states with respect to our assumption:

We define the function $InSt_{\varphi,k}(E_1)$ as the biggest set $I \subset St(\varphi)^k$, such that for every $((q_{11}, \ldots, q_{1l}), \ldots, (q_{k1}, \ldots, q_{kl})) \in I$ the following conditions are satisfied:

For every $j \in [l]$:

- if $x_j \in E_1$, then for every $i \in [k]$, $q_{ij} = d_{x_j}$,
- otherwise, there exists at most one $i \in [k]$ such that $q_{ij} = nd_{x_j}$.

The representation transitions can now be defined formally. We define $dtrans(\varphi)$ as the smallest set by the following scheme:

For every $k \in Rank_\Sigma$:

- case: $k = 0$
  For every $j \in [l]$ and for every $(P_1, P_2) \in CP_\varphi$:
  if $Grp_{\varphi,0}((P_1, P_2)) \neq \emptyset$, then for every $E_1 \in Pr(P_1)$ we have:

  $$(dt_0((Grp_{\varphi,0}((P_1, P_2)), E_1, \{\}), (\,)) = (v_1, \ldots, v_l)) \in dtrans(\varphi),$$

  where for every $j \in [l]$:

  - if $x_j \in E_1$, then $v_j = \varepsilon_{x_j}$,
  - otherwise, $v_j = d_{x_j}$.

- case: $k > 0$
  For every $j \in [l]$ and for every $(P_1, P_2) \in CP_\varphi$:
  if $Grp_{\varphi,k}((P_1, P_2)) \neq \emptyset$, then for every $E_1 \in Pr(P_1)$ and $((q_{11}, \ldots, q_{1l}), \ldots, (q_{k1}, \ldots, q_{kl})) \in InSt_{\varphi,k}(E_1)$ we have:

  $$\begin{aligned}(dt_k(&(Grp_{\varphi,k}((P_1, P_2)), E_1, \{\}),\\ &((q_{11}, \ldots, q_{1l}), \ldots, (q_{k1}, \ldots, q_{kl}))) = (v_1, \ldots, v_l)) \in dtrans(\varphi),\end{aligned}$$

  where for every $j \in [l]$:

  - if $x_j \in E_1$, then $v_j = \varepsilon_{x_j}$,
  - if $\exists! i \in [k].\ q_{ij} = nd_{x_j}$, then $v_j = i.nd_{x_j}$,
  - otherwise, $v_j = d_{x_j}$.

We give the representation transitions of our running example.

Example 9. Consider again our running example:

$$\varphi = gt(\langle often, x\rangle(\langle name, y\rangle), null) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

Then:

$$
\begin{aligned}
& dtrans(\varphi) \\
= \{ \quad & dt_0((\{\mathtt{INTTYP}, \mathtt{BOOLTYP}\}, \{\}, \{\}), (\,)) = (d_x, d_y), \\
& dt_0((\{\mathtt{IDENT}\}, \{y\}, \{\}), (\,)) = (d_x, \varepsilon_y), \\
& dt_0((\{\mathtt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = (d_x, d_y), \\
& dt_1((\{\mathtt{DSINGLE}\}, \{\}, \{\}), ((d_x, d_y))) = (d_x, d_y), \\
& dt_1((\{\mathtt{DSINGLE}\}, \{\}, \{\}), ((d_x, nd_y)) = (d_x, 1.nd_y), \\
& dt_1((\{\mathtt{DSINGLE}\}, \{\}, \{\}), ((nd_x, d_y))) = (1.nd_x, d_y), \\
& dt_1((\{\mathtt{DSINGLE}\}, \{\}, \{\}), ((nd_x, nd_y))) = (1.nd_x, 1.nd_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((d_x, d_y), (d_x, d_y))) = (d_x, d_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((d_x, d_y), (d_x, nd_y))) = (d_x, 2.nd_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((d_x, d_y), (nd_x, d_y))) = (2.nd_x, d_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((d_x, d_y), (nd_x, nd_y))) = (2.nd_x, 2.nd_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((d_x, nd_y), (d_x, d_y))) = (d_x, 1.nd_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((d_x, nd_y), (nd_x, d_y))) = (2.nd_x, 1.nd_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((nd_x, d_y), (d_x, d_y))) = (1.nd_x, d_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((nd_x, d_y), (d_x, nd_y))) = (1.nd_x, 2.nd_y), \\
& dt_2((\{\mathtt{DPAIR}\}, \{\}, \{\}), ((nd_x, nd_y), (d_x, d_y))) = (1.nd_x, 1.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{x\}, \{\}), ((d_x, d_y), (d_x, d_y))) = (\varepsilon_x, d_y), \\
& dt_2((\{\mathtt{DECL}\}, \{x\}, \{\}), ((d_x, d_y), (d_x, nd_y))) = (\varepsilon_x, 2.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{x\}, \{\}), ((d_x, nd_y), (d_x, d_y))) = (\varepsilon_x, 1.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((d_x, d_y), (d_x, d_y))) = (d_x, d_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((d_x, d_y), (d_x, nd_y)) = (d_x, 2.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((d_x, d_y), (nd_x, d_y))) = (2.nd_x, d_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((d_x, d_y), (nd_x, nd_y))) = (2.nd_x, 2.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((d_x, nd_y), (d_x, d_y))) = (d_x, 1.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((d_x, nd_y), (nd_x, d_y))) = (2.nd_x, 1.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((nd_x, d_y), (d_x, d_y))) = (1.nd_x, d_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((nd_x, d_y), (d_x, nd_y))) = (1.nd_x, 2.nd_y), \\
& dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), ((nd_x, nd_y), (d_x, d_y))) = (1.nd_x, 1.nd_y) \quad \}
\end{aligned}
$$

$\square$

As we know from subsection 6.2.2, the automaton of an attribute formula has an infinite number of states. Hence, if we consider a naive construction of transitions we also have an infinite number of transition. We have shown here, that by representing the state symbolically, we are able to obtain a finite number of representation transitions (in our example, we obtain 28 representation transitions).

Remark 1. If an attribute formula becomes a subformula of a built formula, then the value of every node variable in the attribute formula that appears in the input states of a representation transition is represented in the following way:
Let x be a node variable in the attribute formula.

- We use d_x or the general representation of Dewey notation nd_x, if the node variable x is *not* quantified in the built formula where the attribute formula is contained. Every symbolic representation nd_x that appears in the input states represents the same value of the node variable x, since if we let the automaton run on a well-marked tree, there exists exactly one node which is labeled by x (and this node is the value of the node variable x, which is represented by nd_x).
- We use $\emptyset_x$ or $snd_x^{(p,q)}$, if the node variable x is quantified in the built formula where the attribute formula is contained. The additional superscript index (p, q) is uniquely defined for every symbolic representation of a set of nodes for x which occurs in the input states. The additional superscript index (p, q) distinguishes every symbolic representation $snd_x^{(p,q)}$ that appears in the input states. They have to be distinguished, since every such symbolic representation represents different nodes which are guessed for the node variable x.

We define a function which transforms a state containing a symbolic representation of a particular Dewey notation, i.e. the symbolic representation other than nd_x and snd_x, into a symbolic representation representing a general representation, i.e. nd_x and snd_x, and keeps the index x only. This function is needed particularly when the cross product, the projection, and the power set constructions are performed, where we consider only reachable states. When an attribute formula is involved, such reachable state will contain a symbolic representation representing a particular Dewey notation. And since a reachable state is later considered as an input state of a representation transition, then that particular representation of Dewey notation should be transformed into a general one. We define for every representation state of MSO* formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ the transformation function $trf_\varphi : St(\varphi) \to St(\varphi)$ as the following:

- For the formula $\varphi = true$, $\varphi = label_\sigma(x)$, $\varphi = x \in X$, $\varphi = (x == y)$, $\varphi = type_\eta(x)$, $\varphi = edge_i(x, y)$, $\varphi = over(x, y)$:
  $trf_\varphi(q) = q$, with $q \in St(\varphi)$.
- $\varphi \in T^{Bool}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle\cup\mathcal{F}}$:
  $trf_\varphi((v_1, \ldots, v_l)) = (v'_1, \ldots, v'_l)$, where for every $j \in [l]$:
  - $\underline{\text{if}}$ $v_j = \varepsilon_x$, $\underline{\text{then}}$ $v'_j = nd_x$, with $x \in \mathcal{V}_1$,
  - $\underline{\text{if}}$ $v_j = i.nd_x$, $\underline{\text{then}}$ $v'_j = nd_x$, with $i \in \mathbb{N}_+$ and $x \in \mathcal{V}_1$,
  - $\underline{\text{if}}$ $v_j = sc\varepsilon_x$, $\underline{\text{then}}$ $v'_j = snd_x$, with $x \in \mathcal{V}_1$,
  - $\underline{\text{if}}$ $v_j = snd_x^{(p,q)}$, $\underline{\text{then}}$ $v'_j = snd_x$, with $p, q \in \mathbb{N}_+$ and $x \in \mathcal{V}_1$,
  - $\underline{\text{if}}$ $v_j = i.snd_x^{(p,q)}$, $\underline{\text{then}}$ $v'_j = snd_x$, with $i, p, q \in \mathbb{N}_+$ and $x \in \mathcal{V}_1$,
  - $\underline{\text{if}}$ $v_j = sms_x(s_1, s_2)$, $\underline{\text{then}}$ $v'_j = snd_x$, with $x \in \mathcal{V}_1$ and $s_1, s_2 \in Sdw_{\mathcal{V}_1}$,

  – otherwise $v'_j = v_j$.

- $\varphi = \neg\psi$.
  $trf_\varphi(q) = trf_\psi(q)$, with $s \in St(\varphi)$.

- $\varphi = (\psi_1 \wedge \psi_2)$
  $trf_\varphi((q_1, q_2)) = (trf_{\psi_1}(q_1), trf_{\psi_2}(q_2))$, with $q_1 \in St(\psi_1)$ and $q_2 \in St(\psi_2)$.

- $\varphi = (\exists x : \widetilde{\eta}.\ \psi)$.
  $trf_\varphi(P) = \bigcup_{(q,f)\in P}\{(trf_\psi(q), f)\}$, with $P \in St(\varphi)$.

- $\varphi = (\exists X : \widetilde{\eta}.\ \psi)$.
  $trf_\varphi(P) = \bigcup_{q\in P}\{trf_\psi(q)\}$, with $P \in St(\varphi)$.

□

7.3.2 Diamond Operator

To deal with our representation of input symbols in the cross product construction, the projection construction, and the power set construction, we introduce here a binary operator, viz. the *diamond operator*, which is denoted as $\diamond$.

With the diamond operator, given two representation input symbols, e.g. (E', E'_1, E'_2) and (E'', E''_1, E''_2), we compute a new representation input symbol (E, E_1, E_2), which represents *all common input symbols* represented by both (E', E'_1, E'_2) and (E'', E''_1, E''_2). Such common represented input symbols are computed when we need to derive the representation input symbols of the new representation transitions (by one of the above mentioned constructions) from the representation transitions of subformulas (or from the representation of nondeterministic transitions, in case we consider the power set construction).

The idea to compute (E, E_1, E_2) from (E', E'_1, E'_2) and (E'', E''_1, E''_2) is described in the following:

- Since E' (and E'', respectively) is a set of symbols from Σ, then the common symbols of Σ from E' and E'' are simply those symbols which occur in both E' and E''. Hence, E can be computed by taking the *intersection* of E' and E''.

- If E is not empty, then we compute the new set of representation node variables E_1, by taking both the representation node variables in E'_1 and in E'_2. This is because all elements in E are from E' and E'', where the representation input symbol labeled by E' (and E'', respectively) is also labeled by E'_1 (and E''_1, respectively). Since we take both representation node variables from E'_1 and E''_1, we can compute E_1 by taking the *union* of E'_1 and E''_1. One thing we should notice is that the union can not be done, if E'_1 contains a variable without overline, e.g. x, and E''_1 contains the same variable name but with overline, e.g. $\overline{x}$, or vice versa. Intuitively, this means that E'_1 allows x to be included as a part of an input symbol, but on the other hand, E''_1 does not allow x to do so. This contradictory situation disallows the new set of representation node variables E_1 to be computed.

- With the same reason as in the computation of E_1, we compute E_2 by taking the *union* of E_2' and E_2''.

Formally, we define $\diamond : Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \rightarrow Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ as the following: (we write $\diamond$ as an infix operator)

$(E', E_1', E_2') \diamond (E'', E_1'', E_2'') =$

- $\underline{\text{if}}$ $(E' \cap E'' \neq \emptyset)$
  $\wedge\ (\forall x \in E_1'.\ \overline{x} \notin E_1'')\ \wedge\ (\forall x \in E_1''.\ \overline{x} \notin E_1')$
  $\wedge\ (\forall X \in E_2'.\ \overline{X} \notin E_2'')\ \wedge\ (\forall X \in E_2''.\ \overline{X} \notin E_2')$
  $\underline{\text{then}}$: $(E' \cap E'', E_1' \cup E_1'', E_2' \cup E_2'')$,
- $\underline{\text{otherwise}}$: *undefined*.

We give some examples.

Example 10. We compute the common input symbols from the following two representation input symbols by using the diamond operator:

- Consider these two representation input symbols:
  - $(\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\})$ which represents the following input symbols:
    - $(\texttt{IDENT}, \{\}, \{\})$, $(\texttt{IDENT}, \{y\}, \{\})$, $(\texttt{IDENT}, \{z\}, \{\})$, $(\texttt{IDENT}, \{\}, \{X\})$, $(\texttt{IDENT}, \{y\}, \{X\})$, $(\texttt{IDENT}, \{z\}, \{X\})$, $(\texttt{IDENT}, \{y, z\}, \{\})$, $(\texttt{IDENT}, \{y, z\}, \{X\})$,
    - $(\texttt{INTTYP}, \{\}, \{\})$, $(\texttt{INTTYP}, \{\}, \{Y\})$,
    - $(\texttt{BOOLTYP}, \{\}, \{\})$, $(\texttt{BOOLTYP}, \{\}, \{Y\})$,
  - $(\{\texttt{IDENT}\}, \{y\}, \{\})$ which represents the following input symbol:
    - $(\texttt{IDENT}, \{y\}, \{\})$, $(\texttt{IDENT}, \{y\}, \{X\})$, $(\texttt{IDENT}, \{y, z\}, \{\})$, and $(\texttt{IDENT}, \{y, z\}, \{X\})$.

  They have common input symbols, viz.:

  - $(\texttt{IDENT}, \{y\}, \{\})$,
  - $(\texttt{IDENT}, \{y\}, \{X\})$,
  - $(\texttt{IDENT}, \{y, z\}, \{\})$,
  - $(\texttt{IDENT}, \{y, z\}, \{X\})$.

  These common input symbols can be computed by the diamond operator: $(\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}) \diamond (\{\texttt{IDENT}\}, \{y\}, \{\}) = \{\texttt{IDENT}\}, \{y\}, \{\})$, where the result indeed represents those common input symbols.

- Consider these two representation input symbols:
  - $(\{\texttt{DECL}\}, \{x\}, \{\})$ which represents the input symbol $(\texttt{DECL}, \{x\}, \{\})$,
  - $(\{\texttt{DECL}\}, \{\overline{x}\}, \{\})$ which represents the input symbol $(\texttt{DECL}, \{\}, \{\})$.

  Applying the diamond operator to the two given representation input symbols gives the following:
  $(\{\texttt{DECL}\}, \{x\}, \{\}) \diamond (\{\texttt{DECL}\}, \{\overline{x}\}, \{\}) =$ *undefined*, which means there are no common input symbol for these two representation input symbols.

□

7.3.3 Compression of Representation Transitions

In the subsequent subsections, we shall discuss three constructions of representation transitions, viz. the cross product construction, the projection construction, and the power set construction. It turns out that after performing these constructions, we could obtain some representation transitions which have the same behavior but are defined individually. Such representation transitions have the same input states and also give the same output state. They actually differ only in their representation input symbols. Indeed, they can be combined together into one representation transition.

In this subsection we propose a mechanism to compress representation transitions resulting from the three constructions. By compression we mean that we combine some representation transitions which have the same behavior into one representation transition such that eventually after compression we can have less number of representation transitions.

The idea of the compression is the following. Given a set of representation transitions resulting from one of the three constructions, we examine every two representation transitions which have the same input states and the same output state. From their representation input symbols, we can decide whether these two representation transitions can be compressed into one representation transition (which covers exactly both of the compressed representation transitions). Having two representation transitions which have the same input states and the same output states, we combine them if their representation input symbols, say (E', E'_1, E'_2) and (E'', E''_1, E''_2), have one of the following properties:

- The representation input symbols differ only in their first component, i.e. E' and E''. This means we can combine both representation transitions into one representation transition with the representation input symbol $(E' \cup E'', E'_1, E'_2)$, since by our definition of representation input symbol it represents the same input symbols which are represented by both (E', E'_1, E'_2) and (E'', E''_1, E''_2).
- The representation input symbols differ only in their second component, i.e. E'_1 and E''_1. Their difference should be only in one representation node variable, say x, where x exists in E'_1 and $\overline{x}$ exists in E''_1, or vice versa. Intuitively, this means that the occurrence of x in the input symbol does not influence the behavior of the transition. This means we can leave out x from the input symbol. Hence, we can combine both representation transitions into one representation transition with the representation input symbol $(E', E'_1 \setminus \{x\}, E'_2)$ (in case x is in E'_1) or $(E', E'_1 \setminus \{\overline{x}\}, E'_2)$ (in case $\overline{x}$ is in E'_1).
- The representation input symbols differ only in their third component, i.e. E'_2 and E''_2. Similar to the second property, they should only differ in one representation node set variable, say X, where X exists in E'_2 and $\overline{X}$ exists in E''_2, or vice versa. For the similar reason as in the second property, we can combine both representation transitions into one representation transition with the representation input symbol $(E', E'_1, E'_2 \setminus \{X\})$ (in case X is in E'_2) or $(E', E'_1, E'_2 \setminus \{\overline{X}\})$ (in case $\overline{X}$ is in E'_2).

This procedure should be performed to every two different representation transition until there are no more representation transitions which can be compressed.

Given a set of complete and non-overlapping representation transitions with respect to our assumption, we claim that this compression mechanism keeps the set of representation transitions complete and non-overlapping with respect to our assumption. The reason is the following:

- Combining two representation transitions that differ only in their first component of representation input symbols, i.e. E' and E'' in (E', E'_1, E'_2) and (E'', E''_1, E''_2), respectively, keeps the representation transitions completely and non-overlappingly. The reason is that, we combine these two representation transitions only by deriving a new representation input symbol, i.e. $(E' \cup E'', E'_1, E'_2)$ for the new representation transition. But, by our interpretation of representation input symbols (cf. page 37), this new representation input symbol represents exactly the same input symbols as they are represented by (E', E'_1, E'_2) and (E'', E''_1, E''_2).

- Combining two representation transitions that differ only in their second component of representation input symbols, i.e. E'_1 and E''_1 in (E', E'_1, E'_2) and (E'', E''_1, E''_2), respectively, keeps the representation transitions completely and non-overlappingly. The reason is that, we combine these two representation transitions only by deriving a new representation input symbol, where we leave out the representation node variable whose occurrence in the representation input symbol is not significant in determining the output state. But, by our interpretation of representation input symbols (cf. page 37) and our discussion on page 38 (especially in the first item), the new representation input symbol represents exactly the same input symbols as they are represented by (E', E'_1, E'_2) and (E'', E''_1, E''_2).

- For the similar reason as in the previous item, combining two representation transitions that differ only in their third component of representation input symbols, i.e. E'_2 and E''_2 in (E', E'_1, E'_2) and (E'', E''_1, E''_2), respectively, also keeps the representation transitions completely and non-overlappingly.

Based on this idea, we define now the compression of representation transitions formally.

In the following, we assume that two representation transitions with the same input states and the same output state are already given. We define a function that given the representation input symbols of these two representation transitions, it combines them into one representation input symbol, i.e. the representation input symbol of the new compressed representation transition, if they meet one of the above mentioned properties.

We define function $compress : Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \rightarrow Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ as the following:

$compress\ ((E', E'_1, E'_2), (E'', E''_1, E''_2)) =$

- <u>if</u> $((E'_1 = E''_1) \wedge (E'_2 = E''_2))$
  <u>then</u>: $(E' \cup E'', E'_1, E'_2)$

- <u>if</u> $((E' = E'') \wedge (E'_2 = E''_2) \wedge$
  $(\exists E_1 \subseteq \widehat{\mathcal{V}_1}.(E'_1 = E_1 \cup \{x\}) \wedge (E''_1 = E_1 \cup \{\overline{x}\})))$
  <u>then</u>: (E', E_1, E'_2)
- <u>if</u> $((E' = E'') \wedge (E'_1 = E''_1) \wedge$
  $(\exists E_2 \subseteq \widehat{\mathcal{V}_2}.(E'_2 = E_2 \cup \{X\}) \wedge (E''_2 = E_2 \cup \{\overline{X}\})))$
  <u>then</u>: (E', E'_1, E_2)
- <u>otherwise</u>: *undefined*
  (Note: this means the two representation input symbols are not combinable.)

Given a set of representation transitions, every two different representation transitions in this set have to be examined for the possibility of compression, which will produce a new set containing more compact representation transitions (if some representation transitions are successfully compressed). The representation transitions in this new set should again be examined, to see whether the compression procedure can again be applied. This has to be done repeatedly until we obtain the final set of compressed representation transitions, where no more representation transitions in this set can be compressed. The representation transitions which can not be compressed are left unchanged.

We give an example.

Example 11. Consider the following (uncompressed) representation transitions:

1. $dt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\ )) = 0$
2. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0$
3. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1$
4. $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (0, 0)) = 1$
5. $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (0, 1)) = 1$
6. $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (1, 0)) = 1$
7. $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (1, 1)) = 1$
8. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0, 0)) = 0$
9. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0, 1)) = 1$
10. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (1, 0)) = 1$
11. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (1, 1)) = 1$

By comparing every two representation transitions, the following pairs of representation transitions can be compressed:

- representation transition 5 and representation transition 9,
- representation transition 6 and representation transition 10,

- representation transition 7 and representation transition 11,

where the new compressed representation input symbol for each of these pairs is computed as follows:

$$\textit{compress}\ ((\{\texttt{DECL}\},\{\},\{\}),(\{\texttt{DPAIR}\},\{\},\{\})) = (\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}).$$

Note that the rest of the representation transitions can not be compressed. Hence, after compressing the three pairs of representation transitions we have the following:

1. $dt_0((\{\texttt{IDENT},\texttt{INTTYP},\texttt{BOOLTYP}\},\{\},\{\}),(\ )) = 0$
2. $dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),(0)) = 0$
3. $dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),(1)) = 1$
4. $dt_2((\{\texttt{DECL}\},\{\},\{\}),(0,0)) = 1$
5. $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(0,0)) = 0$
6. $dt_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),(0,1)) = 1$
7. $dt_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),(1,0)) = 1$
8. $dt_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),(1,1)) = 1$

From the last result, we compare again every two representation transitions to see whether the compression procedure can be applied again. Since no more representation transitions can be compressed, then this eight representation transitions is the final result. By this example, we can see that the compression procedure is able to minimize the number of representation transitions from 11 representation transitions to 8 representation transitions. □

7.3.4 Representation Transitions by Cross Product Construction

In this subsection, we show how the cross product construction is realized for our representation transitions. Recall that the cross product construction is needed when we deal with a built formula of MSO* logic using conjunction. In the following, first we shall give the idea of the construction informally and then its formal construction.

We compute the cross product of the representation transitions by using the reachability construction. This means we compute the representation transitions only for the reachable states. Thus, we can decrease the number of representation transitions.

In the reachability construction, we maintain four sets, i.e. the sets T_{cr}, S_{cr}, ΔT_{cr}, and ΔS_{cr}. The set S_{cr} contains all the states that are reachable so far, whereas the set ΔS_{cr} contains the new reachable states resulting from the new representation transitions with all possible current reachable states as their input states. The content of the set S_{cr} is then constantly updated by the set ΔS_{cr}. Hence, the set S_{cr} can be seen as an accumulator for the reachable states.

The set T_{cr} contains all representation transitions of the cross product construction so far. It is also updated constantly with the new transition functions which are computed in ΔT_{cr}. Hence, the set T_{cr} can also be seen as an accumulator for the representation transitions.

The set S_{cr} and the set T_{cr} are initially empty. The set ΔT_{cr} initially contains the new representation transitions of all possible representation input symbols with the rank 0 and the set ΔS_{cr} initially contains all reachable states (output states) of the new representation transitions in the initial ΔT_{cr}.

The reachability construction is completed whenever no new reachable states are produced anymore, i.e. when $\Delta S_{cr} = \emptyset$. At this point the current set S contains all (reachable) states and the current set T_{cr} together with the current set ΔT_{cr} contain all representation transitions of the cross product construction.

To compute the new representation transitions, in general we use the same approach as in the original cross product construction (cf. subsection 6.2.2 for the automaton of formula with conjunction). The particular things of our cross product construction (compared to the original construction) are that we only consider reachable states (instead of all states) and we have representation input symbols (instead of the original input symbols). To derive the representation input symbol of the new representation transition, we use the diamond operator.

As a running example, consider the following formula:

$$(label_{\mathtt{DECL}}(x) \wedge edge_1(x,y)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}},$$

and for the purpose of the following presentation, suppose that we are now computing the representation transitions of this formula for the rank 2 with the input states ((0,1),(0,0)), where both states (0,1) and (0,0) are reachable states.

We give in the following the idea of how to compute a new representation transition of the formula $(\psi_1 \wedge \psi_2)$, for the rank k with $k \in Rank_\Sigma$, given a k-tuple of reachable states, say $((q_{11}, q_{21}), \ldots, (q_{1k}, q_{2k}))$, as the input states for the new representation transition (recall that an input state of a new transition is a pair of states, where the first element of the pair is a state of ψ_1, whereas the second one is a state of ψ_2):

1. We extract from $((q_{11}, q_{21}), \ldots, (q_{1k}, q_{2k}))$, two k-tuples of states, viz. $(q_{11}, \ldots, q_{1k})$ and $(q_{21}, \ldots, q_{2k})$.
   For our running example, we have the following two 2-tuples of states: (0,0) and (1,0) which are obtained from ((0,1),(0,0)).

2. Since the idea of the cross product construction is to let the automata run parallel on a tree, we search from the set of representation transitions of ψ_1, the representation transitions with $(q_{11}, \ldots, q_{1k})$ as their input states (for short, let us call them $reps_1$) and from the set of representation transitions of ψ_2, the representation transitions with $(q_{21}, \ldots, q_{2k})$ as their input states (for short, let us call them $reps_2$), if they exist. If they do not exist, then it means that no representation transition with the k-tuple of input states $((q_{11}, q_{21}), \ldots, (q_{1k}, q_{2k}))$ is needed, since this configuration of input states can never occur in the parallel run of the automata for the subformulas.

   For our running example, we have the following representation transitions of the formula $label_{\mathtt{DECL}}(x)$ with the input state (0,0), cf. example 3:

- $dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (0,0)) = 1$
- $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0,0)) = 0$
- $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,0)) = 0$

and the following representation transitions of the formula $edge_1(x, y)$ with the input state (1,0), cf. example 7:

- $dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (1,0)) = 2$
- $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (1,0)) = 0$

3. By using the representation input symbols from $reps_1$ and $reps_2$, we derive the representation input symbols for the new representation transitions. Since the new representation input symbols are used for the new representation transitions with the input states $((q_{11}, q_{21}), \ldots, (q_{1k}, q_{2k}))$, then this new representation input symbols should represents the common represented input symbols both from $reps_1$ and $reps_2$. These common represented input symbols can be computed via the diamond operator by taking each representation input symbol of $reps_1$ and each representation input symbol of $reps_2$ as the arguments of the diamond operator.

   For our running example, we compute the following:

   - $(\{\texttt{DECL}\}, \{x\}, \{\}) \diamond (\{\texttt{DECL}\}, \{x\}, \{\}) = (\{\texttt{DECL}\}, \{x\}, \{\})$,
   - $(\{\texttt{DECL}\}, \{x\}, \{\}) \diamond (\{\texttt{DECL}\}, \{\overline{x}\}, \{\}) = undefined$ (or, no common represented input symbol),
   - $(\{\texttt{DECL}\}, \{\overline{x}\}, \{\}) \diamond (\{\texttt{DECL}\}, \{x\}, \{\}) = undefined$ (or, no common represented input symbol),
   - $(\{\texttt{DECL}\}, \{\overline{x}\}, \{\}) \diamond (\{\texttt{DECL}\}, \{\overline{x}\}, \{\}) = (\{\texttt{DECL}\}, \{\overline{x}\}, \{\})$,
   - $(\{\texttt{DPAIR}\}, \{\}, \{\}) \diamond (\{\texttt{DECL}\}, \{x\}, \{\}) = undefined$ (or, no common represented input symbol),
   - $(\{\texttt{DPAIR}\}, \{\}, \{\}) \diamond (\{\texttt{DECL}\}, \{\overline{x}\}, \{\}) = undefined$ (or, no common represented input symbol).

   For this example, we only have two new representation input symbols: $(\{\texttt{DECL}\}, \{x\}, \{\})$ and $(\{\texttt{DECL}\}, \{\overline{x}\}, \{\})$, each for the representation transition with the configuration input states ((0,1),(0,0)). Note that we do not obtain a new representation input symbol with the first component containing `DPAIR`, since if we consider the run of the automaton for the formula $edge_i(x, y)$, it will not happen that a node which is labeled by `DPAIR` has a first successor node which is labeled by y (this is reflected by the state (1,0), which we consider for the automaton of the formula $edge_i(x, y)$). The reason is that y has the sort {`Ident`}, whereas the first successor of the node labeled by `DPAIR` has the sort `Decl`. And since the run of the automaton for $label_{\texttt{DECL}}(x) \wedge edge_i(x, y)$ on a tree t means that both the automaton for $label_{\texttt{DECL}}(x)$ and the automaton for $edge_i(x, y)$ run parallel on the same tree t, then there will be also no such tree t on which the automaton for $label_{\texttt{DECL}}(x) \wedge edge_i(x, y)$ runs. Hence, no representation transition with the representation input symbol containing `DPAIR` is needed.

Remark 2. With respect to our assumption, given that the representation input symbols of $reps_1$ represent all input symbols for the input states $(q_{11}, \ldots, q_{1k})$ and the representation input symbols of $reps_2$ represent all input symbols for the input states $(q_{21}, \ldots, q_{2k})$, we can claim that by using our diamond operator we derive the new representation input symbols that represent all input symbols for the input states $((q_{11}, q_{21}), \ldots, (q_{1k}, q_{2k}))$, with respect to our assumption. This is because the new representation input symbols are derived by considering each representation input symbols of $reps_1$ and each representation input symbols of $reps_2$ as the arguments of the diamond operator when we compute the new representation input symbols. $\square$

4. The output state of a new representation transition, which is a pair of states, is derived by pairing the output states of the representation transitions of the subformulas, from which their representation input symbols are derived.

   For our running example, we obtain the following two representation transitions (for the configuration input states ((0,1),(0,0)) and two new representation input symbols):

   - $dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), ((0,1),(0,0))) = (1,2)$
   - $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), ((0,1),(0,0))) = (0,0)$

With respect to our assumption, we claim that the new representation transitions produced by our cross product construction indeed represents the transitions of the automaton for a built formula with conjunction (as described in subsection 6.2.2), since:

- in general, to produce a new representation transition we use the same approach as in the original cross product construction, which produce the transitions of the automaton for a built formula with conjunction. Though our construction only considers reachable states, but the essence of the construction is still the same.

- though we use our representation input symbols in the representation transitions, in Remark 2, we have argued that for a given configuration of input states we can compute the representation input symbols that represents all input symbols for this configuration of input states. Since we can do this for every possible configuration of input states, then with respect to our assumption, this cross product construction will compute complete and non-overlapping representation transitions.

Based on this idea, we define now the reachability construction for cross product formally.

To compute the set of representation transitions $dtrans(\psi_1 \wedge \psi_2)$ of the automaton for the formula $(\psi_1 \wedge \psi_2) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, we assume that the set of representation transitions $dtrans(\psi_1)$ of the automaton for the formula $\psi_1 \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ and the set of representation transitions $dtrans(\psi_2)$ of the automaton for the formula $\psi_2 \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ are given.

First, we define for a given rank k with $k \in Rank_{\Sigma,+}$, the function $crtrans_k : \mathcal{P}(St(\psi_1 \wedge \psi_2))^k \rightarrow \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$ that given k sets of reachable states, it computes the new representation transitions of k-tuples of input states, where each k-tuple of input states comes from the given k sets of reachable states:

$$\begin{array}{ll} & crtrans_k(S_1, \ldots, S_k) \\ = \{ & dt_k((E, E_1, E_2), ((q'_1, q''_1), \ldots, (q'_k, q''_k))) = (q', q'') \mid \\ & (E, E_1, E_2) = (E', E'_1, E'_2) \diamond (E'', E''_1, E''_2), \\ & \text{where } (E, E_1, E_2) \text{ is not } \textit{undefined}, \\ & (q'_1, q''_1) \in S_1, \ldots, (q'_k, q''_k) \in S_k, \\ & (dt_k((E', E'_1, E'_2), (q'_1, \ldots, q'_k)) = q') \in dtrans(\psi_1), \\ & (dt_k((E'', E''_1, E''_2), (q''_1, \ldots, q''_k)) = q'') \in dtrans(\psi_2), \\ & (E', E'_1, E'_2), (E'', E''_1, E''_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2), \\ & q' \in St(\psi_1), q'' \in St(\psi_2) \qquad \} \end{array}$$

We define the reachability construction for cross product as function $reach_{cr} : \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St(\psi_1 \wedge \psi_2)) \times \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St(\psi_1 \wedge \psi_2)) \rightarrow \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St(\psi_1 \wedge \psi_2))$. It is defined based on the idea of the "semi-naive" method [14] for computing the closure of a given set (in this case, the set of reachable states) according to an operation (in this case, the cross product construction) by keeping track of just the truly new data items (in this case, the new reachable states) which are added to the set on each pass.
The function $reach_{cr}$ is defined as the following:

$reach_{cr}\ (T_{cr}, S_{cr}, \Delta T_{cr}, \Delta S_{cr}) =$

- if $\Delta S_{cr} = \emptyset$,
  then: $(T_{cr} \cup \Delta T_{cr}, S_{cr})$
- otherwise: $reach_{cr}\ (T'_{cr}, S'_{cr}, \Delta T_{cr_{>0}}, \Delta S_{cr_{>0}})$,

where:

- $T'_{cr} = T_{cr} \cup \Delta T_{cr}$
  Note: T'_{cr} collects all representation transitions computed so far.
- $S'_{cr} = S_{cr} \cup \Delta S_{cr}$
  Note: S'_{cr} collects all states which are reachable so far.
- $$\Delta T_{cr_{>0}} = \bigcup_{k \in Rank_{\Sigma,+}} \Big(\begin{array}{l} crtrans_k(\underbrace{S_{cr}, \ldots, S_{cr}}_{k-1}, \Delta S_{cr}) \cup \\ crtrans_k(\underbrace{S_{cr}, \ldots, S_{cr}}_{k-2}, \Delta S_{cr}, S'_{cr}) \cup \ldots \cup \\ crtrans_k(\Delta S_{cr}, \underbrace{S'_{cr}, \ldots, S'_{cr}}_{k-1}) \end{array} \Big)$$

  Note: $\Delta T_{cr_{>0}}$ computes for every rank $k \in Rank_{\Sigma,+}$ the new representation transitions, which is performed by function $crtrans_k$. The new representation transitions are only computed for the states reachable so far as their input states. By "semi-naive" method, we avoid to compute for every rank $k \in Rank_{\Sigma,+}$, $crtrans_k(\underbrace{S_{cr}, \ldots, S_{cr}}_{k})$, since this means we compute the representation transitions considering only the old reachable states, but this in fact has been computed in the previous recursive call.

- $$\begin{aligned}\Delta S_{cr_{>0}} = \{ \quad & trf_{(\psi_1 \wedge \psi_2)}(q) \in St(\psi_1 \wedge \psi_2) \mid \\ & (dt_k((E, E_1, E_2), (q_1, \dots, q_k)) = q) \in \Delta T_{cr_{>0}}, \\ & k > 0,\ (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2), \\ & q_1, \dots, q_k \in S'_{cr} \qquad \} \setminus S'_{cr}\end{aligned}$$

  Note: $\Delta S_{cr_{>0}}$ collects the new reachable states, with function $trf_{(\psi_1 \wedge \psi_2)}$ defined in remark 1.

We can compute now the set of representation transitions $dtrans(\psi_1 \wedge \psi_2)$ of the automaton for the formula $(\psi_1 \wedge \psi_2) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ by using the reachability construction for cross product.

Let $(trans_{cr}, sts_{cr}) = reach_{cr}\ (\emptyset, \emptyset, \Delta T_{cr=0}, \Delta S_{cr=0})$, where:

$$\begin{aligned}\Delta T_{cr=0} = \{ \quad & dt_0((E, E_1, E_2), (\,)) = (q', q'') \mid \\ & (E, E_1, E_2) = (E', E'_1, E'_2) \diamond (E'', E''_1, E''_2), \\ & \text{where } (E, E_1, E_2) \text{ is not } \textit{undefined}, \\ & (dt_0((E', E'_1, E'_2), (\,)) = q') \in dtrans(\psi_1), \\ & (dt_0((E'', E''_1, E''_2), (\,)) = q'') \in dtrans(\psi_2), \\ & (E', E'_1, E'_2), (E'', E''_1, E''_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2), \\ & q' \in St(\psi_1), q'' \in St(\psi_2) \qquad \}\end{aligned}$$

$$\begin{aligned}\Delta S_{cr=0} = \{ \quad & trf_{(\psi_1 \wedge \psi_2)}(q) \in St(\psi_1 \wedge \psi_2) \mid \\ & (dt_0((E, E_1, E_2), (\,)) = q) \in \Delta T_{cr=0}, \\ & (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \qquad \}\end{aligned}$$

Then, we have:

$$dtrans(\psi_1 \wedge \psi_2) = trans_{cr}.$$

Additionally, sts_{cr} is the set of all reachable states of the automaton for the formula $(\psi_1 \wedge \psi_2) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$.

Example 12. Consider again our running example:

$$(label_{\texttt{DECL}}(x) \wedge edge_1(x, y)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

Let us recall the representation transitions $dtrans(\psi_1)$ of the automaton for the subformula $\psi_1 = label_{\texttt{DECL}}(x)$ which was given in example 3:

1. $dt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0$,
2. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0$,
3. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1$,
4. $dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (0, 0)) = 1$,
5. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0, 0)) = 0$,
6. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (1, 0)) = 1$,
7. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0, 1)) = 1$,

8. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,0)) = 0$,
9. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (1,0)) = 1$,
10. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,1)) = 1$.

and the representation transitions $dtrans(\psi_2)$ of the automaton for the subformula $\psi_2 = edge_1(x, y)$ which was given in example 7:

1. $dt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) = 1$,
2. $dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = 0$,
3. $dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0$,
4. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0$,
5. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (2)) = 2$,
6. $dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0,0)) = 0$,
7. $dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (1,0)) = 2$,
8. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (1,0)) = 0$,
9. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (2,0)) = 2$,
10. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0,2)) = 2$,
11. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (2,0)) = 2$,
12. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0,2)) = 2$.

We compute the set of representation transitions $dtrans(\psi_1 \wedge \psi_2)$ from $dtrans(\psi_1)$ and $dtrans(\psi_2)$ by using the reachability construction of cross product. We start by computing:

$$reach_{cr}\ (\emptyset, \emptyset, \Delta T_{cr=0}, \Delta S_{cr=0}),$$

where:

$$\begin{array}{lcll}
\Delta T_{cr=0} & = \{ & dt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) = (0,1), & \\
 & & dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = (0,0), & \\
 & & dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = (0,0) & \} \\
 & & & \\
\Delta S_{cr=0} & = & \{(0,1), (0,0)\} & .
\end{array}$$

Following our formal definition of $reach_{cr}$, we perform the following recursive call (we give here an additional index to the arguments, just to distinguish the arguments of one recursive call with the other recursive call):

$$reach_{cr}\ (T'_{cr_{(1)}}, S'_{cr_{(1)}}, \Delta T_{cr_{>0(1)}}, \Delta S_{cr_{>0(1)}}),$$

where:

$$\begin{array}{llll}
T'_{cr_{(1)}} & = & & \Delta T_{cr_{=0}} \\
S'_{cr_{(1)}} & = & & \Delta S_{cr_{=0}} \\
 & = & & \{(0,1),(0,0)\} \\
\Delta T_{cr_{>0(1)}} & = & & crtrans_1(\Delta S_{cr_{=0}})\cup \\
 & & & crtrans_2(\emptyset,\Delta S_{cr_{=0}}) \cup crtrans_2(\Delta S_{cr_{=0}}, S'_{cr_{(1)}}) \\
 & = & & crtrans_1(\Delta S_{cr_{=0}}) \cup \emptyset \cup crtrans_2(\Delta S_{cr_{=0}}, S'_{cr_{(1)}}) \\
 & = & & crtrans_1(\Delta S_{cr_{=0}}) \cup crtrans_2(\Delta S_{cr_{=0}}, S'_{cr_{(1)}}) \\
 & = & \{ & dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),((0,0))) = (0,0) \quad \} \\
 & & \cup & \\
 & & \{ & dt_2((\{\texttt{DECL}\},\{x\},\{\}),((0,0),(0,0))) = (1,0), \\
 & & & dt_2((\{\texttt{DECL}\},\{\overline{x}\},\{\}),((0,0),(0,0))) = (0,0), \\
 & & & dt_2((\{\texttt{DPAIR}\},\{\},\{\}),((0,0),(0,0))) = (0,0), \\
 & & & dt_2((\{\texttt{DECL}\},\{x\},\{\}),((0,1),(0,0))) = (1,2), \\
 & & & dt_2((\{\texttt{DECL}\},\{\overline{x}\},\{\}),((0,1),(0,0))) = (0,0) \quad \} \\
\Delta S_{cr_{>0(1)}} & = & & \{(0,0),(1,0),(1,2)\} \setminus S'_{cr_{(1)}} \\
 & = & & \{(1,0),(1,2)\}
\end{array}$$

Since $\Delta S_{cr_{>0(1)}} \neq \emptyset$, we perform the following recursive call:

$$reach_{cr}\ (T'_{cr_{(2)}}, S'_{cr_{(2)}}, \Delta T_{cr_{>0(2)}}, \Delta S_{cr_{>0(2)}}),$$

where:

$$\begin{array}{llll}
T'_{cr_{(2)}} & = & & T'_{cr_{(1)}} \cup \Delta T_{cr_{>0(1)}} \\
S'_{cr_{(2)}} & = & & S'_{cr_{(1)}} \cup \Delta S_{cr_{>0(1)}} \\
 & = & & \{(0,1),(0,0),(1,0),(1,2)\} \\
\Delta T_{cr_{>0(2)}} & = & & crtrans_1(\Delta S_{cr_{>0(1)}})\cup \\
 & & & crtrans_2(S'_{cr_{(1)}}, \Delta S_{cr_{>0(1)}})\cup \\
 & & & crtrans_2(\Delta S_{cr_{>0(1)}}, S'_{cr_{(2)}}) \\
 & = & \{ & dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),((1,0))) = (1,0), \\
 & & & dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),((1,2))) = (1,2) \quad \} \\
 & & \cup & \\
 & & \{ & dt_2((\{\texttt{DECL}\},\{\overline{x}\},\{\}),((0,0),(1,0))) = (1,0), \\
 & & & dt_2((\{\texttt{DPAIR}\},\{\},\{\}),((0,0),(1,0))) = (1,0), \\
 & & & dt_2((\{\texttt{DECL}\},\{\overline{x}\},\{\}),((0,0),(1,2))) = (1,2), \\
 & & & dt_2((\{\texttt{DPAIR}\},\{\},\{\}),((0,0),(1,2))) = (1,2), \\
 & & & dt_2((\{\texttt{DECL}\},\{\overline{x}\},\{\}),((0,1),(1,0))) = (1,0) \quad \} \\
 & & \cup & \\
 & & \{ & dt_2((\{\texttt{DECL}\},\{\overline{x}\},\{\}),((1,0),(0,0))) = (1,0), \\
 & & & dt_2((\{\texttt{DPAIR}\},\{\},\{\}),((1,0),(0,0))) = (1,0), \\
 & & & dt_2((\{\texttt{DECL}\},\{\overline{x}\},\{\}),((1,2),(0,0))) = (1,2), \\
 & & & dt_2((\{\texttt{DPAIR}\},\{\},\{\}),((1,2),(0,0))) = (1,2) \quad \} \\
\Delta S_{cr_{>0(2)}} & = & & \{(1,0),(1,2)\} \setminus S'_{cr_{(2)}} \\
 & = & & \emptyset
\end{array}$$

Since $\Delta S_{cr_{>0(2)}} = \emptyset$, we have then the following:

$$reach_{cr}\ (T'_{cr_{(2)}}, S'_{cr_{(2)}}, \Delta T_{cr_{>0(2)}}, \Delta S_{cr_{>0(2)}}) = (T'_{cr_{(2)}} \cup \Delta T_{cr_{>0(2)}}, S'_{cr_{(2)}})$$

and the representation transitions $dtrans(\varphi)$ of the automaton for the formula $\varphi = label_{\texttt{DECL}}(x) \wedge edge_1(x,y)$ resulting from the cross product construction is the following:

$$
\begin{array}{rll}
dtrans(\varphi) & = & T'_{cr_{(2)}} \cup \Delta T_{cr_{>0(2)}} \\
& = \{ & dt_0((\{\mathtt{IDENT}\},\{y\},\{\}),(\,)) = (0,1), \\
& & dt_0((\{\mathtt{IDENT}\},\{\overline{y}\},\{\}),(\,)) = (0,0), \\
& & dt_0((\{\mathtt{INTTYP},\mathtt{BOOLTYP}\},\{\},\{\}),(\,)) = (0,0), \\
& & dt_1((\{\mathtt{DSINGLE}\},\{\},\{\}),((0,0))) = (0,0), \\
& & dt_1((\{\mathtt{DSINGLE}\},\{\},\{\}),((1,0))) = (1,0), \\
& & dt_1((\{\mathtt{DSINGLE}\},\{\},\{\}),((1,2))) = (1,2), \\
& & dt_2((\{\mathtt{DECL}\},\{x\},\{\}),((0,0),(0,0))) = (1,0), \\
& & dt_2((\{\mathtt{DECL}\},\{\overline{x}\},\{\}),((0,0),(0,0))) = (0,0), \\
& & dt_2((\{\mathtt{DECL}\},\{x\},\{\}),((0,1),(0,0))) = (1,2), \\
& & dt_2((\{\mathtt{DECL}\},\{\overline{x}\},\{\}),((0,1),(0,0))) = (0,0), \\
& & dt_2((\{\mathtt{DECL}\},\{\overline{x}\},\{\}),((0,0),(1,0))) = (1,0), \\
& & dt_2((\{\mathtt{DECL}\},\{\overline{x}\},\{\}),((0,0),(1,2))) = (1,2), \\
& & dt_2((\{\mathtt{DECL}\},\{\overline{x}\},\{\}),((0,1),(1,0))) = (1,0), \\
& & dt_2((\{\mathtt{DECL}\},\{\overline{x}\},\{\}),((1,0),(0,0))) = (1,0), \\
& & dt_2((\{\mathtt{DECL}\},\{\overline{x}\},\{\}),((1,2),(0,0))) = (1,2), \\
& & dt_2((\{\mathtt{DPAIR}\},\{\},\{\}),((0,0),(0,0))) = (0,0), \\
& & dt_2((\{\mathtt{DPAIR}\},\{\},\{\}),((0,0),(1,0))) = (1,0), \\
& & dt_2((\{\mathtt{DPAIR}\},\{\},\{\}),((0,0),(1,2))) = (1,2), \\
& & dt_2((\{\mathtt{DPAIR}\},\{\},\{\}),((1,0),(0,0))) = (1,0), \\
& & dt_2((\{\mathtt{DPAIR}\},\{\},\{\}),((1,2),(0,0))) = (1,2) \quad \}
\end{array}
$$

Additionally, $S'_{cr_{(2)}} = \{(0,1),(0,0),(1,0),(1,2)\}$ is the set of all reachable states of the automaton for the formula $label_{\mathtt{DECL}}(x) \wedge edge_1(x,y)$.

If we attempt to apply the compression procedure on $dtrans(\varphi)$, we shall get the same result, because no representation transitions in $dtrans(\varphi)$ can be more compressed. □

We give a comparison of the number of states and transitions using our representation and those which are constructed by following the formal construction given in subsection 6.2.2. This is shown in table 7.9.

	Non-represented	Represented
Number of States	6	4
Number of Transition	126	20

Table 7.9: A comparison of the number of states and transitions (represented and non-represented) for the formulas $label_{\mathtt{DECL}}(x) \wedge edge_1(x,y)$

Note that the following states are not reachable:

- (0,2), since the second component of this pair (considering the second subformula, $edge_1(x,y)$) means that the nodes labeled by y and x have been visited (hence, we have the state 2), but on the other hand the first component of this pair (considering the first subformula, $label_{\mathtt{DECL}}(x)$)

means that the node labeled by x has never been visited (hence, we have the state 0). Thus, such combination of states can not occur.

- (1,1). The second component of this pair (considering the second subformula, $edge_1(x, y)$) means that a node which is labeled by y is visited. With respect to Σ and the sort of y ({`Ident`}), only the nodes which are labeled by `IDENT` can also be labeled by y, since `IDENT` is the only symbol of Σ which has the result sort `Ident`. On the other hand, the first component of this pair (considering the first subformula, $label_{\texttt{DECL}}(x)$) means that the node labeled by x has been visited. But, this can not be true, since the node which is labeled by `IDENT` and y is a leaf (`IDENT` is a nullary symbol of Σ), hence no other node has been visited before (in particular, also the node which is labeled by x). Moreover, the node which is labeled by `IDENT` can not be labeled by x, since x has the sort {`Decl`}, which does not fit to the result sort of `IDENT`.

Those 126 transitions in table 7.9 are obtained by following the formal construction of the automata for formulas with conjunction as given in subsection 6.2.2, where every input symbol and every state as the input states of the transitions are considered:

- 8 transitions having the input symbols with the first component: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, where we consider 2 node variables ($y^{\{\texttt{Ident}\}}$ and $z^{\{\texttt{Ident}\}}$), 1 node set variable ($X^{\{\texttt{Ident}\}}$), and no input state,

- 2 transitions having the input symbols with the first component: $\texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,

- 2 transitions having the input symbols with the first component: $\texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,

- 6 transitions having the input symbols with the first component: $\texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}$, where we consider no node variables, no node set variable, and 6 combinations of input states, i.e. ((0,0)), ((0,1)), ((0,2)), ((1,0)), ((1,1)), ((1,2)),

- 36 transitions having the input symbols with the first component: $\texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}$, where we consider no node variable, no node set variable, and 36 combinations of input states,

- 72 transitions having the input symbols with the first component: $\texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}$, where we consider 1 node variable ($x^{\{\texttt{Decl}\}}$), no node set variable, and 36 combinations of input states.

7.3.5 Representation Transitions by Projection Construction

In this subsection, we show how the projection construction is realized for our representation transitions. Recall that the projection construction is needed when we deal with a built formula of MSO* logic using quantifier, where we obtain a nondeterministic tree automaton from a deterministic one.

As in the cross product construction, we compute the projection of the representation transitions by using the reachability construction. This means we compute the representation transitions only for the reachable states. Thus, we can decrease the number of representation transitions.

The idea of the reachability construction of projection is the same as in the cross product. We do not repeat it here. The difference is only when computing the new representation transitions, which we shall discuss later. As in the reachability construction of cross product, for projection we also maintain four sets, i.e. the sets T_{pr}, S_{pr}, ΔT_{pr}, and ΔS_{pr}. Each of them has the same role as its counterpart in the cross product construction (cf. the roles of the sets T_{cr}, S_{cr}, ΔT_{cr}, and ΔS_{cr} in subsection 7.3.4).

Before we give the idea of how to compute the representation transitions by the reachability construction of projection, we need to define our representation transition of a nondeterministic bottom-up tree automaton.

We define the function $ndtrans : MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}} \rightarrow \mathcal{P}(Ntr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$, which takes a formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, where:

- $\varphi = (\exists x : \widetilde{\eta}.\ \psi)$, or
- $\varphi = (\exists X : \widetilde{\eta}.\ \psi)$.

and gives a set $ndtrans(\varphi) \in \mathcal{P}(Ntr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$ of representation transitions for the nondeterministic bottom-up tree automaton of the formula φ, where $Ntr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ is a set of representation transitions, such that for every (rank) $k \in \mathbb{N}$, $\varphi = (\exists x : \overline{\eta}.\ \psi)$ or $\varphi = (\exists X : \overline{\eta}.\ \psi)$, and $ndt_{k,\varphi} \in Ntr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$:

$$ndt_{k,\varphi} : Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St_{non}(\varphi)^k \rightarrow \mathcal{P}(St_{non}(\varphi)).$$

If φ is clear from the context, we write ndt_k instead of $ndt_{k,\varphi}$.

Basically, the formula with quantifier over node variable and over node set variable have the same pattern of reachability construction. The difference is only in computing the new representation transitions.
We define the reachability construction for projection as the following function $reach_{pr} : \mathcal{P}(Ntr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St_{non}(\varphi)) \times \mathcal{P}(Ntr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St_{non}(\varphi)) \rightarrow \mathcal{P}(Ntr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St_{non}(\varphi))$:

$reach_{pr}\ (T_{pr}, S_{pr}, \Delta T_{pr}, \Delta S_{pr}) =$

- <u>if</u> $\Delta S_{pr} = \emptyset$,
  <u>then</u>: $(T_{pr} \cup \Delta T_{pr}, S_{pr})$
- <u>otherwise</u>: $reach_{pr}\ (T'_{pr}, S'_{pr}, \Delta T_{pr_{>0}}, \Delta S_{pr_{>0}})$

where:

- $T'_{pr} = T_{pr} \cup \Delta T_{pr}$
  Note: T'_{pr} collects all representation transitions computed so far.

- $S'_{pr} = S_{pr} \cup \Delta S_{pr}$
  Note: S'_{pr} collects all states which are reachable so far.

- $$\Delta T_{pr_{>0}} = \begin{cases} \Delta T_{pr1_{>0}} & \text{if } \varphi = \exists x : \widetilde{\eta}.\ \psi \\ \Delta T_{pr2_{>0}} & \text{if } \varphi = \exists X : \widetilde{\eta}.\ \psi \end{cases}$$

  Note: $\Delta T_{pr_{>0}}$ computes for every rank $k \in Rank_{\Sigma,+}$ the new representation transitions. For the first order quantification formula, it is performed by the function $\Delta T_{pr1_{>0}}$, whereas for the second order quantification formula by $\Delta T_{pr2_{>0}}$. We shall define them, when we discuss each of these quantification formulas. As in the cross product construction, the new representation transitions are only computed for the states reachable so far as their input states.

- $$\begin{aligned} \Delta S_{pr_{>0}} \quad = \{ \quad & q \in trf_\varphi(P) \mid \\ & (ndt_k((E, E_1, E_2), (q_1, \ldots, q_k)) = P) \in \Delta T_{pr_{>0}}, \\ & k > 0,\ (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2), \\ & q_1, \ldots, q_k \in S'_{pr},\ P \in \mathcal{P}(St_{non}(\varphi)) \qquad \} \setminus S'_{pr} \end{aligned}$$

  Note: $\Delta S_{pr_{>0}}$ collects the new reachable states, with function trf_φ defined in remark 1.

The set of representation transitions $ndtrans(\varphi)$ of the automaton for the formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ can be computed as the following.
Let $(trans_{pr}, sts_{pr}) = reach_{pr}\ (\emptyset, \emptyset, \Delta T_{pr_{=0}}, \Delta S_{pr_{=0}})$, where:

- $$\Delta T_{pr_{=0}} = \begin{cases} \Delta T_{pr1_{=0}} & \text{if } \varphi = \exists x : \widetilde{\eta}.\ \psi \\ \Delta T_{pr2_{=0}} & \text{if } \varphi = \exists X : \widetilde{\eta}.\ \psi \end{cases}$$

  Note: The functions $\Delta T_{pr1_{=0}}$ and $\Delta T_{pr2_{=0}}$ will be defined later when we discuss each of the quantification formula.

- $$\begin{aligned} \Delta S_{pr_{=0}} \quad =\{ \quad & q \in trf_\varphi(P) \mid \\ & (ndt_0((E, E_1, E_2), (\,)) = P) \in \Delta T_{pr_{=0}}, \\ & (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2),\ P \in \mathcal{P}(St_{non}(\varphi)) \quad \} \end{aligned}$$

Then, we have:

$$ndtrans(\varphi) = trans_{pr}.$$

Additionally, sts_{pr} is the set of all reachable states of the automaton for the formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$.

To compute the new representation transitions (in $\Delta T_{pr_{=0}}$ and $\Delta T_{pr_{>0}}$), generally we use the same approach as in the original projection construction (cf. subsection 6.2.2 for the automaton of formula with quantifier). The particular things of our projection construction (compared to the original construction) are that we consider reachable states only (instead of all states) and we have representation input symbols (instead of the original input symbols). To derive the representation input symbol of the new representation transition, we also use the diamond operator.

We give in the following the idea of how to compute the representation transitions of the formula $\exists x : \widetilde{\eta}.\ \psi$ and the formula $\exists X : \widetilde{\eta}.\ \psi$, for the rank k with $k \in Rank_\Sigma$.

$\boxed{\varphi = \exists x : \widetilde{\eta}.\ \psi}$

To compute the representation transitions $ndtrans(\varphi)$ for the formula φ, we assume that the set of the representation transitions $dtrans(\psi)$ for the formula $\psi \in MSO^*(\Sigma, \mathcal{V}_1 \cup \{x^{\widetilde{\eta}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ is given.

As in the original projection construction, we compute a new transition by considering two transitions from the subformula which only differ in the input symbol, where one transition has x in its input symbol and the other does not. For the projection construction in our representation, in general we follow the original projection construction and proceed as the following (by case distinction):

- We start by looking for a representation transition from $dtrans(\psi)$ with the representation input symbol containing $\overline{x}$, which means that x is not contained in the input symbol of the transition represented. Following the original projection construction, we should find the representation transition with the representation input symbol containing x and having the *common represented input symbols* modulo the quantified variable x (the common represented input symbols can be computed by our diamond operator). Here we can have two possibilities:
  - We successfully find such representation transition. Then, a new representation transition is constructed from the representation transition with $\overline{x}$ in the input symbol and the representation transition with x in the input symbol. The new representation transition reflects that we may do both the guessing and no guessing of x.
  - We do not find such representation transition. This is possible, since if a representation transition reflects that a node labeled by x has already been visited, then due to our assumption we need only to have a representation transition with the representation input symbol containing $\overline{x}$. For instance, for the formula $label_{\mathtt{DECL}}(x)$ we have:

    $$dt_2((\{\mathtt{DECL}\}, \{\overline{x}\}, \{\}), (1, 0)) = 1,$$

    and we do not have:

    $$dt_2((\{\mathtt{DECL}\}, \{x\}, \{\}), (1, 0)) = 1,$$

    In this case, a new representation is only constructed from the representation transition with $\overline{x}$ in the input symbol, which reflects that only no guessing of x can be made.
- We have a representation transition from $dtrans(\psi)$ with the representation input symbol containing neither $\overline{x}$ nor x. Such representation transition can exist if the output state of the representation transition does not depend on the fact whether x occurs or $\overline{x}$ occurs in the representation input symbol. In this situation, in case no guessing of x has been

made before, the new representation transitions are defined by splitting the representation input symbol into the following:

- A representation transition with the representation input symbol containing those Σ-symbols whose result sort is in the sort of x. In this case, we may do both the guessing of x or no guessing.
- A representation transition with the representation input symbol containing those Σ-symbols whose result sort is not in the sort of x. In this case, only no guessing of x can be made.

Additionally, in case we have done a guessing of x before, we do not need to split the representation input symbol (with respect to the sort of x), since no more guessing is needed.

We define now formally how we compute the new representation transitions based on the idea above.

First, we define a function that collects from $dtrans(\psi)$ the representation transitions for the rank $k \in Rank_\Sigma$ with the representation input symbol containing the given node variable and the given input states, $coltr1_k : \mathcal{V}_1 \times St(\psi)^k \to \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$. It is defined as follows:

$$\begin{array}{rl} & coltr1_k(x, (q_1, \dots, q_k)) \\ = \{ & (dt_k((E, E_1, E_2), (q_1, \dots, q_k)) = q) \in dtrans(\psi) \mid \\ & (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2),\ x \in E_1,\ q \in St(\psi) \quad \} \end{array}$$

The set of representation transitions for φ are computed as follows. We define the set $\Delta T_{pr1_{=0}}$ which contains the representation transitions of φ for the rank 0 and the set $\Delta T_{pr1_{>0}}$ which contains the representation transitions of φ for the rank greater than 0. Recall that both sets are used in the reachability construction of projection, i.e. $reach_{pr}$.

1. $\Delta T_{pr1_{=0}}$ is defined as the smallest set in the following scheme:
   For every $(dt_0((E, E_1, E_2), (\ )) = q) \in dtrans(\psi)$, with $(E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ and $q \in St(\psi)$:

   - Case 1: $\overline{x} \in E_1$
     - if $coltr1_0(x, (\ )) \neq \emptyset$, then:
       for every $(dt_0((E', E'_1, E'_2), (\ )) = q') \in coltr1_0(x, (\ ))$:
       if $(E, E_1 \setminus \{\overline{x}\}, E_2) \diamond (E', E'_1 \setminus \{x\}, E'_2)$ is defined and $(E'', E''_1, E''_2) = (E, E_1 \setminus \{\overline{x}\}, E_2) \diamond (E', E'_1 \setminus \{x\}, E'_2)$, then:
       $$(ndt_0((E'', E''_1, E''_2), (\ )) = \{(q, \bot), (q', \top)\}) \in \Delta T_{pr1_{=0}}$$
     - otherwise:
       $$(ndt_0((E, E_1 \setminus \{\overline{x}\}, E_2), (\ )) = \{(q, \bot)\}) \in \Delta T_{pr1_{=0}}$$
   - Case 2: $x, \overline{x} \notin E_1$
     - if there is $\sigma \in E$ with $sort_{res}(\sigma) \in \widetilde{\eta}$, then:
       Let $elem_{\sigma,\widetilde{\eta}} = \{\sigma \in E \mid sort_{res}(\sigma) \in \widetilde{\eta}\}$.
       $(ndt_0((elem_{\sigma,\widetilde{\eta}}, E_1, E_2), (\ )) = \{(q, \bot), (q, \top)\}) \in \Delta T_{pr1_{=0}}$

    - <u>if</u> there is $\sigma \in E$ with $sort_{res}(\sigma) \notin \widetilde{\eta}$, <u>then</u>:
      Let $nelem_{\sigma,\widetilde{\eta}} = \{\sigma \in E \mid sort_{res}(\sigma) \notin \widetilde{\eta}\}$.
      $(ndt_0((nelem_{\sigma,\widetilde{\eta}}, E_1, E_2), (\,)) = \{(q, \bot)\}) \in \Delta T_{pr1_{=0}}$

2. $\Delta T_{pr1_{>0}}$ is defined as the smallest set in the following scheme:

   For every $(dt_k((E, E_1, E_2), (q_1, \ldots, q_k)) = q) \in dtrans(\psi)$, with:
   $k \in Rank_{\Sigma,+}$, $(E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, and $q, q_1, \ldots, q_k \in St(\psi)$:

   - <u>Case 1</u>: $\overline{x} \in E_1$
     - <u>if</u> $coltr1_k(x, (q_1, \ldots, q_k)) \neq \emptyset$, <u>then</u>:
       Let $ct_1 = coltr1_k(x, (q_1, \ldots, q_k))$.
       For every $(dt_k((E', E'_1, E'_2), (q_1, \ldots, q_k)) = q') \in ct_1$:
       <u>if</u> $(E, E_1 \setminus \{\overline{x}\}, E_2) \diamond (E', E'_1 \setminus \{x\}, E'_2)$ is defined and
       $(E'', E''_1, E''_2) = (E, E_1 \setminus \{\overline{x}\}, E_2) \diamond (E', E'_1 \setminus \{x\}, E'_2)$, <u>then</u>:
       * <u>if</u> $(q_1, \bot), \ldots, (q_k, \bot) \in S'_{pr}$, <u>then</u>:
         $$(ndt_k((E'', E''_1, E''_2), ((q_1, \bot), \ldots, (q_k, \bot))) \\ = \{(q, \bot), (q', \top)\}) \in \Delta T_{pr1_{>0}}$$
       * for every $i \in [k]$:
         <u>if</u> $(q_1, \bot), \ldots, (q_{i-1}, \bot), (q_i, \top), (q_{i+1}, \bot), \ldots, (q_k, \bot) \in S'_{pr}$, <u>then</u>:
         $$(ndt_k((E'', E''_1, E''_2), ((q_1, \bot), \ldots, (q_{i-1}, \bot), (q_i, \top), \\ (q_{i+1}, \bot), \ldots, (q_k, \bot))) = \{(q, \top)\}) \in \Delta T_{pr1_{>0}}.$$
     - <u>otherwise</u>:
       * <u>if</u> $(q_1, \bot), \ldots, (q_k, \bot) \in S'_{pr}$, <u>then</u>:
         $$(ndt_k((E, E_1 \setminus \{\overline{x}\}, E_2), \\ ((q_1, \bot), \ldots, (q_k, \bot))) = \{(q, \bot)\}) \in \Delta T_{pr1_{>0}}$$
       * for every $i \in [k]$:
         <u>if</u> $(q_1, \bot), \ldots, (q_{i-1}, \bot), (q_i, \top), (q_{i+1}, \bot), \ldots, (q_k, \bot) \in S'_{pr}$, <u>then</u>:
         $$(ndt_k((E, E_1 \setminus \{\overline{x}\}, E_2), ((q_1, \bot), \ldots, (q_{i-1}, \bot), (q_i, \top), \\ (q_{i+1}, \bot), \ldots, (q_k, \bot))) = \{(q, \top)\}) \in \Delta T_{pr1_{>0}}.$$
   - <u>Case 2</u>: $x, \overline{x} \notin E_1$
     - <u>if</u> there is $\sigma \in E$ with $sort_{res}(\sigma) \in \widetilde{\eta}$ and
       $(q_1, \bot), \ldots, (q_k, \bot) \in S'_{pr}$, <u>then</u>:
       Let $elem_{\sigma,\widetilde{\eta}} = \{\sigma \in E \mid sort_{res}(\sigma) \in \widetilde{\eta}\}$.
       $$(ndt_k(elem_{\sigma,\widetilde{\eta}}, E_1, E_2), \\ ((q_1, \bot), \ldots, (q_k, \bot))) = \{(q, \bot), (q, \top)\}) \in \Delta T_{pr1_{>0}}$$

– if there is $\sigma \in E$ with $sort_{res}(\sigma) \notin \widetilde{\eta}$ and $(q_1, \bot), \ldots, (q_k, \bot) \in S'_{pr}$, then:
Let $nelem_{\sigma,\widetilde{\eta}} = \{\sigma \in E \mid sort_{res}(\sigma) \notin \widetilde{\eta}\}$.

$$(ndt_k((nelem_{\sigma,\widetilde{\eta}}, E_1, E_2), ((q_1, \bot), \ldots, (q_k, \bot))) = \{(q, \bot)\}) \in \Delta T_{pr1_{>0}}$$

Additionally, for every $i \in [k]$:
if $(q_1, \bot), \ldots, (q_{i-1}, \bot), (q_i, \top), (q_{i+1}, \bot), \ldots, (q_k, \bot) \in S'_{pr}$, then:

$$(ndt_k((E, E_1, E_2), ((q_1, \bot), \ldots, (q_{i-1}, \bot), (q_i, \top), (q_{i+1}, \bot), \ldots, (q_k, \bot))) = \{(q, \top)\}) \in \Delta T_{pr1_{>0}}.$$

We give an example.

Example 13. Consider the following formula:

$$\varphi = (\exists x : \{\texttt{Decl}\}.\ label_{\texttt{DECL}}(x)) \in MSO^*(\Sigma, \mathcal{V}_1 \setminus \{x^{\{\texttt{Decl}\}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$$

.
The representation transitions for the subformula:

$$label_{\texttt{DECL}}(x) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$$

was already given in the example 3, i.e the following:

1. $dt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0$,
2. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0$,
3. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1$,
4. $dt_2((\{\texttt{DECL}\}, \{x\}, \{\}), (0, 0)) = 1$,
5. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0, 0)) = 0$,
6. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (1, 0)) = 1$,
7. $dt_2((\{\texttt{DECL}\}, \{\overline{x}\}, \{\}), (0, 1)) = 1$,
8. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0, 0)) = 0$,
9. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (1, 0)) = 1$,
10. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (0, 1)) = 1$.

We compute the set of representation transitions $ndtrans(\varphi)$ from $dtrans(\psi)$ by using the reachability construction of projection. In the following, we put $(\star)$ after a representation transition to mean that the representation transition is derived by case 1 and $(\star\star)$ for the representation transition which is derived by case 2. We start by computing:

$$reach_{pr}\ (\emptyset, \emptyset, \Delta T_{pr_{=0}}, \Delta S_{pr_{=0}}),$$

where:

$$\begin{array}{lll}
\Delta T_{pr_{=0}} & = & \Delta T_{pr1_{=0}} \\
 & = & \{ndt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = \{(0, \bot)\}^{(\star\star)}\} \\
\Delta S_{pr_{=0}} & = & \{(0, \bot)\}.
\end{array}$$

Following our formal definition of $reach_{pr}$, we perform the following recursive call (we give here an additional index to the arguments, just to distinguish the arguments of one recursive call with the other recursive call):

$$reach_{pr}\ (T'_{pr_{(1)}}, S'_{pr_{(1)}}, \Delta T_{pr_{>0(1)}}, \Delta S_{pr_{>0(1)}}),$$

where:

$$\begin{array}{lll}
T'_{pr_{(1)}} & = & \Delta T_{pr_{=0}} \\
S'_{pr_{(1)}} & = & \Delta S_{pr_{=0}} \\
 & = & \{(0, \bot)\} \\
\Delta T_{pr_{>0(1)}} & = & \Delta T_{pr1_{>0(1)}} \\
 & = \{ & ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), ((0, \bot))) = \{(0, \bot)\}^{(\star\star)}, \\
 & & ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), ((0, \bot), (0, \bot))) \\
 & & \qquad = \{(0, \bot), (1, \top)\}^{(\star)}, \\
 & & ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), ((0, \bot), (0, \bot))) = \{(0, \bot)\}^{(\star\star)} \quad \} \\
\Delta S_{pr_{>0(1)}} & = & \{(0, \bot), (1, \top)\} \setminus S'_{pr_{(1)}} \\
 & = & \{(1, \top)\}
\end{array}$$

Since $\Delta S_{pr_{>0(1)}} \neq \emptyset$, we perform the following recursive call:

$$reach_{pr}\ (T'_{pr_{(2)}}, S'_{pr_{(2)}}, \Delta T_{pr_{>0(2)}}, \Delta S_{pr_{>0(2)}}),$$

where:

$$\begin{array}{lll}
T'_{pr_{(2)}} & = & T'_{pr_{(1)}} \cup \Delta T_{pr_{>0(1)}} \\
S'_{pr_{(2)}} & = & S'_{pr_{(1)}} \cup \Delta S_{pr_{>0(1)}} \\
 & = & \{(0, \bot), (1, \top)\} \\
\Delta T_{pr_{>0(2)}} & = & \Delta T_{pr1_{>0(2)}} \\
 & = & \Delta T_{pr1_{>0(1)}} \quad \cup \\
 & \{ & ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), ((1, \top))) = \{(1, \top)\}^{(\star\star)}, \\
 & & ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), ((1, \top), (0, \bot))) = \{(1, \top)\}^{(\star)}, \\
 & & ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), ((0, \bot), (1, \top))) = \{(1, \top)\}^{(\star)}, \\
 & & ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), ((1, \top), (0, \bot))) = \{(1, \top)\}^{(\star\star)}, \\
 & & ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), ((0, \bot), (1, \top))) = \{(1, \top)\}^{(\star\star)} \quad \} \\
\Delta S_{pr_{>0(2)}} & = & \{(1, \top)\} \setminus S'_{pr_{(2)}} \\
 & = & \emptyset
\end{array}$$

Since $\Delta S_{pr_{>0(2)}} = \emptyset$, we have then the following:

$$reach_{pr}\ (T'_{pr_{(2)}}, S'_{pr_{(2)}}, \Delta T_{pr_{>0(2)}}, \Delta S_{pr_{>0(2)}}) = (T'_{pr_{(2)}} \cup \Delta T_{pr_{>0(2)}}, S'_{pr_{(2)}})$$

and the representation transitions $ndtrans(\varphi)$ of the automaton for the formula $\varphi = \exists x : \{\texttt{Decl}\}.\ label_{\texttt{DECL}}(x)$ resulting from the projection construction is the following:

$$\begin{array}{ll}
 & \mathit{ndtrans}(\varphi) \\
= & T'_{pr_{(2)}} \cup \Delta T_{pr_{>0(2)}} \\
= \{ & \mathit{ndt}_0((\{\texttt{IDENT},\texttt{INTTYP},\texttt{BOOLTYP}\},\{\},\{\}),(\,)) = \{(0,\bot)\}, \\
 & \mathit{ndt}_1((\{\texttt{DSINGLE}\},\{\},\{\}),((0,\bot))) = \{(0,\bot)\}, \\
 & \mathit{ndt}_1((\{\texttt{DSINGLE}\},\{\},\{\}),((1,\top))) = \{(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DECL}\},\{\},\{\}),((0,\bot),(0,\bot))) = \{(0,\bot),(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DECL}\},\{\},\{\}),((1,\top),(0,\bot))) = \{(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DECL}\},\{\},\{\}),((0,\bot),(1,\top))) = \{(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DPAIR}\},\{\},\{\}),((0,\bot),(0,\bot))) = \{(0,\bot)\}, \\
 & \mathit{ndt}_2((\{\texttt{DPAIR}\},\{\},\{\}),((1,\top),(0,\bot))) = \{(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DPAIR}\},\{\},\{\}),((0,\bot),(1,\top))) = \{(1,\top)\} \qquad \}
\end{array}$$

Additionally, $S'_{pr_{(2)}} = \{(0,\bot),(1,\top)\}$ is the set of all reachable states of the automaton for the formula $\exists x : \{\texttt{Decl}\}.\ \mathit{label}_{\texttt{DECL}}(x)$.

If we attempt to apply the compression procedure on $\mathit{ndtrans}(\varphi)$, it turns out that some representation transitions can be compressed:

- $\mathit{ndt}_2((\{\texttt{DECL}\},\{\},\{\}),((1,\top),(0,\bot))) = \{(1,\top)\}$ and $\mathit{ndt}_2((\{\texttt{DPAIR}\},\{\},\{\}),((1,\top),(0,\bot))) = \{(1,\top)\}$,
- $\mathit{ndt}_2((\{\texttt{DECL}\},\{\},\{\}),((0,\bot),(1,\top))) = \{(1,\top)\}$ and $\mathit{ndt}_2((\{\texttt{DPAIR}\},\{\},\{\}),((0,\bot),(1,\top))) = \{(1,\top)\}$

where the representation input symbol of every compressed representation transition from these two pairs is computed as the following:

$$\mathit{compress}\ ((\{\texttt{DECL}\},\{\},\{\}),(\{\texttt{DPAIR}\},\{\},\{\})) = (\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}).$$

Hence, after compressing the two pairs of representation transitions we have the following (which is also the final result, since no more representation transitions can be compressed):

$$\begin{array}{ll}
 & \mathit{ndtrans}(\varphi) \\
= \{ & \mathit{ndt}_0((\{\texttt{IDENT},\texttt{INTTYP},\texttt{BOOLTYP}\},\{\},\{\}),(\,)) = \{(0,\bot)\}, \\
 & \mathit{ndt}_1((\{\texttt{DSINGLE}\},\{\},\{\}),((0,\bot))) = \{(0,\bot)\}, \\
 & \mathit{ndt}_1((\{\texttt{DSINGLE}\},\{\},\{\}),((1,\top))) = \{(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DECL}\},\{\},\{\}),((0,\bot),(0,\bot))) = \{(0,\bot),(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DPAIR}\},\{\},\{\}),((0,\bot),(0,\bot))) = \{(0,\bot)\}, \\
 & \mathit{ndt}_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),((1,\top),(0,\bot))) = \{(1,\top)\}, \\
 & \mathit{ndt}_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),((0,\bot),(1,\top))) = \{(1,\top)\} \qquad \}
\end{array}$$

□

We give a comparison of the number of states and transitions using this representation and those which are constructed by following the formal construction given in subsection 6.2.2. This is shown in table 7.10.

Note that the following states are not reachable:

- $(0,\top)$, since this state means that the guessing of x is made (shown by the second component of the pair, $\top$) at the node where the guessing actually can not be made (shown by the first component of the pair, 0).

	Non-represented	Represented
Number of States	4	2
Number of Transition	48	7

Table 7.10: A comparison of the number of states and transitions (represented and non-represented) for the formulas $\exists x : \{\texttt{Decl}\}.\ \mathit{label}_{\texttt{DECL}}(x)$

- $(1, \bot)$, since this state means that the guessing of x is not made (shown by the second component of the pair, $\bot$) at the node where the guessing actually should be made (shown by the first component of the pair, 1).

Those 48 transitions in table 7.10 are obtained by following the formal construction of the automata for formulas with quantifier over node variable as given in subsection 6.2.2, where every input symbol and every state as the input states of the transitions are considered:

- 8 transitions having the input symbols with the first component: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, where we consider 2 node variables ($y^{\{\texttt{Ident}\}}$ and $z^{\{\texttt{Ident}\}}$), 1 node set variable ($X^{\{\texttt{Ident}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 4 transitions having the input symbols with the first component: $\texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}$, where we consider no node variables, no node set variable, and 4 combinations of input states, i.e. $((0,\bot))$, $((0,\top))$, $((1,\bot))$, and $((1,\top))$,
- 16 transitions having the input symbols with the first component: $\texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}$, where we consider no node variable, no node set variable, and 16 combinations of input states,
- 16 transitions having the input symbols with the first component: $\texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}$, where we consider no node variable (since x is the quantified variable), no node set variable, and 16 combinations of input states.

$\boxed{\varphi = \exists X : \widetilde{\eta}.\ \psi}$

To compute the representation transitions $\mathit{ndtrans}(\varphi)$ for the formula φ, we assume that the set of representation transitions $\mathit{dtrans}(\psi)$ for the formula $\psi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2 \cup \{X^{\widetilde{\eta}}\})_{\mathcal{A},\mathcal{F}}$ is given.

Computing the new representation transitions for a second order quantification formula is simpler than for the first order quantification formula. The reason is that the well-marked trees on which we assume the automaton runs do not restrict the number of guessings of the quantified variable X (in contrast to the guessing of a node variable x). In general, to construct a new representation transition, we still follow the original projection construction for the second order quantification formula and proceed as the following (by case distinction):

- We start by looking for a representation transition from $dtrans(\psi)$ with the representation input symbol containing $\overline{X}$, which means that X is not contained in the input symbol of the transition represented. Following the original projection construction, we should find the representation transition with the representation input symbol containing X and having the *common represented input symbols* modulo the quantified variable X (the common represented input symbols can be computed by our diamond operator). In contrast to the first order quantification, here we have only one possibility:

  – we successfully find such representation transition and a new representation transition is constructed from the representation transition with $\overline{X}$ in the input symbol and the representation transition with X in the input symbol which reflects that we may do both the guessing of X or no guessing.

  Note that given a representation transition with the input symbol containing $\overline{X}$ we can always successfully find a representation transition with the input symbol containing X. The reason is that we never use a second order variable with overline, in our representation input symbol to restrict that only one node in a well-marked tree can be labeled by X.

- We have a representation transition from $dtrans(\psi)$ with the representation input symbol containing neither $\overline{X}$ nor X. Such representation transition can exist if the occurrence of X in the representation input symbol of this representation transition is not significant to determine its output state. In this case, we do not split the representation input symbol (with respect to the sort of X) as in the first order quantification case, since the output state is always the same, either:

  – we make a guessing or no guessing at the node whose sort is in the sort of X, or

  – we make no guessing at the node whose sort is not in the sort of X.

We define now formally how we compute the new representation transitions based on the idea above.

First, we define a function that collects from $dtrans(\psi)$ the representation transitions for the rank $k \in Rank_\Sigma$ with the representation input symbol containing the given node set variable and the given input states, $coltr2_k : \mathcal{V}_2 \times St(\psi)^k \rightarrow \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$. It is defined as follows:

$$\begin{array}{ll} & coltr2_k(X, (q_1, \ldots, q_k)) \\ = \{ & (dt_k((E, E_1, E_2), (q_1, \ldots, q_k)) = q) \in dtrans(\psi) \mid \\ & (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2),\ X \in E_2,\ q \in St(\psi) \quad \} \end{array}$$

The set of representation transitions for φ are computed as follows. We define the set $\Delta T_{pr2_{=0}}$ which contains the representation transitions of φ for the rank 0 and the set $\Delta T_{pr2_{>0}}$ which contains the representation transitions of φ for the rank greater than 0. Recall that both sets are used in the reachability construction of projection, i.e. $reach_{pr}$.

1. $\Delta T_{pr2_{=0}}$ is defined as the smallest set in the following scheme:
   For every $(dt_0((E, E_1, E_2), (\,)) = q) \in dtrans(\psi)$, with $(E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$ and $q \in St(\psi)$:

   - Case 1: $\overline{X} \in E_2$
     - if $coltr2_0(X, (\,)) \neq \emptyset$, then:
       for every $(dt_0((E', E'_1, E'_2), (\,)) = q') \in coltr2_0(x, (\,))$:
       if $(E, E_1, E_2 \setminus \{\overline{X}\}) \diamond (E', E'_1, E'_2 \setminus \{X\})$ is defined and $(E'', E''_1, E''_2) = (E, E_1, E_2 \setminus \{\overline{X}\}) \diamond (E', E'_1, E'_2 \setminus \{X\})$, then:
       $$(ndt_0((E'', E''_1, E''_2), (\,)) = \{q, q'\}) \in \Delta T_{pr2_{=0}}$$
   - Case 2: $X, \overline{X} \notin E_2$
     Then:
     $$(ndt_0((E, E_1, E_2), (\,)) = \{q\}) \in \Delta T_{pr2_{=0}}$$

2. $\Delta T_{pr2_{>0}}$ is defined as the smallest set in the following scheme:

   For every $(dt_k((E, E_1, E_2), (q_1, \ldots, q_k)) = q) \in dtrans(\psi)$, with:
   $k \in Rank_{\Sigma,+}$, $(E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2)$, and $q, q_1, \ldots, q_k \in St(\psi)$:

   - Case 1: $\overline{X} \in E_2$
     - if $coltr2_k(X, (q_1, \ldots, q_k)) \neq \emptyset$, then:
       Let $ct_2 = coltr2_k(X, (q_1, \ldots, q_k))$.
       For every $(dt_k((E', E'_1, E'_2), (q_1, \ldots, q_k)) = q') \in ct_2$:
       if $(E, E_1, E_2 \setminus \{\overline{X}\}) \diamond (E', E'_1, E'_2 \setminus \{X\})$ is defined, where $(E'', E''_1, E''_2) = (E, E_1, E_2 \setminus \{\overline{X}\}) \diamond (E', E'_1, E'_2 \setminus \{X\})$ and $q_1, \ldots, q_k \in S'_{pr}$, then:
       $$(ndt_k((E'', E''_1, E''_2), (q_1, \ldots, q_k)) = \{q, q'\}) \in \Delta T_{pr2_{>0}}$$
   - Case 2: $X, \overline{X} \notin E_2$
     - if $q_1, \ldots, q_k \in S'_{pr}$, then:
       $$(ndt_k((E, E_1, E_2), (q_1, \ldots, q_k)) = \{q\}) \in \Delta T_{pr2_{>0}}$$

We give an example.

Example 14. Consider the following formula:

$$\varphi = (\exists X : \{\texttt{Ident}\}.\ y \in X) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2 \setminus \{X^{\{\texttt{Ident}\}}\})_{\mathcal{A},\mathcal{F}}.$$

The representation transitions for the subformula:

$$(y \in X) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$$

was already given in the example 4, i.e the following:

1. $dt_0((\{\texttt{IDENT}\}, \{y\}, \{X\}), (\,)) = 1$
2. $dt_0((\{\texttt{IDENT}\}, \{y\}, \{\overline{X}\}), (\,)) = 0$
3. $dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = 0$
4. $dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = 0$
5. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = 0$
6. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = 1$
7. $dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 0)) = 0$
8. $dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (1, 0)) = 1$
9. $dt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 1)) = 1$

We compute the set of representation transitions $ndtrans(\varphi)$ from $dtrans(\psi)$ by using the reachability construction of projection. As in the example 13, we also put $(\star)$ after a representation transition to mean that the representation transition is derived by case 1 and $(\star\star)$ for the representation transition which is derived by case 2. We start by computing:

$$reach_{pr}\ (\emptyset, \emptyset, \Delta T_{pr_{=0}}, \Delta S_{pr_{=0}}),$$

where:

$$\begin{array}{lcll}
\Delta T_{pr_{=0}} & = & \Delta T_{pr2_{=0}} & \\
 & = \{ & ndt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) = \{0, 1\}^{(\star)}, & \\
 & & ndt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = \{0\}^{(\star\star)}, & \\
 & & ndt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = \{0\}^{(\star\star)} & \} \\
 & & & \\
\Delta S_{pr_{=0}} & = & \{0, 1\} &
\end{array}$$

Following our formal definition of $reach_{pr}$, we perform the following recursive call (we give here an additional index to the arguments, just to distinguish the arguments of one recursive call with the other recursive call):

$$reach_{pr}\ (T'_{pr_{(1)}}, S'_{pr_{(1)}}, \Delta T_{pr_{>0(1)}}, \Delta S_{pr_{>0(1)}}),$$

where:

$$\begin{array}{lcll}
T'_{pr_{(1)}} & = & \Delta T_{pr_{=0}} & \\
S'_{pr_{(1)}} & = & \Delta S_{pr_{=0}} & \\
 & = & \{0, 1\} & \\
\Delta T_{pr_{>0(1)}} & = & \Delta T_{pr2_{>0(1)}} & \\
 & = \{ & ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = \{0\}^{(\star\star)}, & \\
 & & ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = \{1\}^{(\star\star)}, & \\
 & & ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 0)) = \{0\}^{(\star\star)}, & \\
 & & ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (1, 0)) = \{1\}^{(\star\star)}, & \\
 & & ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 1)) = \{1\}^{(\star\star)} & \} \\
 & & & \\
\Delta S_{pr_{>0(1)}} & = & \{0, 1\} \setminus S'_{pr_{(1)}} & \\
 & = & \emptyset &
\end{array}$$

Since $\Delta S_{pr_{>0(1)}} = \emptyset$, we have then the following:

$$reach_{pr}\ (T'_{pr_{(1)}}, S'_{pr_{(1)}}, \Delta T_{pr_{>0(1)}}, \Delta S_{pr_{>0(1)}}) = (T'_{pr_{(1)}} \cup \Delta T_{pr_{>0(1)}}, S'_{pr_{(1)}})$$

and the representation transitions $ndtrans(\varphi)$ of the automaton for the formula $\varphi = \exists X : \{\texttt{Ident}\}.\ y \in X$ is the following:

$$\begin{array}{rl} ndtrans(\varphi) = & T'_{pr_{(1)}} \cup \Delta T_{pr_{>0(1)}} \\ \{ & ndt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) = \{0, 1\}, \\ & ndt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = \{0\}, \\ & ndt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = \{0\}, \\ & ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (0)) = \{0\}, \\ & ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (1)) = \{1\}, \\ & ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 0)) = \{0\}, \\ & ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (1, 0)) = \{1\}, \\ & ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), (0, 1)) = \{1\} \qquad \} \end{array}$$

Additionally, $S'_{pr_{(1)}} = \{0, 1\}$ is the set of all reachable states of the automaton for the formula $\exists X : \{\texttt{Ident}\}.\ y \in X$.

If we attempt to apply the compression procedure on $ndtrans(\varphi)$, we shall get the same result, because no representation transitions in $ndtrans(\varphi)$ can be more compressed. □

We give a comparison of the number of states and transitions using this representation and those which are constructed by following the formal construction given in subsection 6.2.2. This is shown in table 7.11.

	Non-represented	Represented
Number of States	2	2
Number of Transition	22	8

Table 7.11: A comparison of the number of states and transitions (represented and non-represented) for the formulas $\exists X : \{\texttt{Ident}\}.\ y \in X$

Those 22 transitions in table 7.11 are obtained by following the formal construction of the automata for formulas with quantifier over node set variable as given in subsection 6.2.2, where every input symbol and every state as the input states of the transitions are considered:

- 4 transitions having the input symbols with the first component: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, where we consider 2 node variables ($y^{\{\texttt{Ident}\}}$ and $z^{\{\texttt{Ident}\}}$), no node set variable (since X is the quantified variable), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,

- 2 transitions having the input symbols with the first component: $\texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}$, where we consider no node variables, no node set variable, and 2 combinations of input states, i.e. ((0)) and ((1)),
- 4 transitions having the input symbols with the first component: $\texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}$, where we consider no node variable, no node set variable, and 4 combinations of input states,
- 8 transitions having the input symbols with the first component: $\texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}$, where we consider 1 node variable ($x^{\{\texttt{Decl}\}}$), no node set variable, and 4 combinations of input states.

With respect to our assumption, we claim that the new representation transitions produced by our projection construction indeed represent the transitions of the automaton for a built formula with existential quantifier (as described in subsection 6.2.2), since:

- in general, we use the same approach as in the original projection construction to produce a new representation transition, except that we consider only reachable states, but the essence of the construction is still the same.
- to define the representation input symbol of a new representation transition, we consider the common represented input symbols which are computed by the diamond operator. Given that $dtrans(\psi)$ is complete and non-overlapping, we shall also obtain the complete and non-overlapping $ndtrans(\varphi)$, since to produce a representation transition in $ndtrans(\varphi)$, our construction considers all possible cases of representation input symbols (regarding the occurrence of the quantified variable) in the representation transitions of $dtrans(\psi)$.

7.3.6 Representation Transitions by Power Set Construction

In this subsection, we show how the power set construction is realized for our representation transitions. Recall that the power set construction is needed to determinize the nondeterministic bottom-up tree automaton resulting from the projection on a first or second order variable.

Generally we use the power set construction defined in section 5.3 to produce the new transitions in our representation. Particularly we compute the new representation transitions by considering only the reachable states as the input states. In this way, we can have less number of representation transitions compared to the number of transitions in case we consider all states.

The idea of the reachability construction of power set is the same as in the cross product construction. We do not repeat it here. The difference is only when computing the new representation transitions, which we shall discuss later. As in the reachability construction of cross product, for the power set construction we also maintain four sets, i.e. the sets T_{ps}, S_{ps}, ΔT_{ps}, and ΔS_{ps}. Each of them has the same role as its counterpart in the cross product construction (cf. the roles of the sets T_{cr}, S_{cr}, ΔT_{cr}, and ΔS_{cr} in subsection 7.3.4).

As a running example for our presentation, consider that we want to determinize the representation transitions of the formula:

$$\exists x : \{\texttt{Decl}\}.\, label_{\texttt{DECL}}(x),$$

whose representation transitions (nondeterministic) are given in the example 13. For our presentation, suppose that we are now computing the representation transitions (deterministic) of this formula for the rank 2 with the reachable states $\{(0, \bot)\}$ and $\{(0, \bot), (1, \top)\}$ as the input states, i.e.

$$(\{(0, \bot)\}, \{(0, \bot), (1, \top)\})$$

(recall from section 5.3, that the state of a deterministic automaton by the power set construction is a set).

Based on the original power set construction, we give now the idea of how to compute a new representation deterministic transition. Note that a special handling is needed when we use our procedure of power set construction to compute the representation deterministic transitions for the first order quantification formula, in particular if the subformula contains an attribute formula and we quantify over a node variable that occurs in the attribute formula. In our following discussion, we first give a construction which works for quantification formula where no attribute formula is involved in the subformula. Later, we show the problem that occurs if an attribute formula is involved in the subformula and we quantify over a node variable that occurs in the attribute formula. We propose then a solution to overcome this problem, where we use a symbolic representation of a set of nodes which we introduced in section 7.2.

Given a k-tuple of reachable states, say $(P_1, \ldots, P_k)$, with $k \in Rank_\Sigma$, as the input states for the new representation transition, to compute a new representation transition, we need to decide the *representation input symbol* and the *output state*. We compute them as the following:

1. According to the original construction, the output state of a deterministic transition for a particular input symbol with the input states $(P_1, \ldots, P_k)$ is the union of all output states of the nondeterministic transitions for the same input symbol with the input states $(q_1, \ldots, q_k) \in P_1 \times \ldots \times P_k$. Based on this, we proceed as the following. We collect all representation nondeterministic transitions with the input states $(q_1, \ldots, q_k) \in P_1 \times \ldots \times P_k$. Note that for some input states $(q_1, \ldots, q_k) \in P_1 \times \ldots \times P_k$, the representation nondeterministic transitions may not exist, for the reason that such input states are not reachable. Thus, we actually collect only those representation nondeterministic transitions with the *reachable* input states $(q_1, \ldots, q_k) \in P_1 \times \ldots \times P_k$.

   For our running example, we collect the representation transitions with the input state $(q_1, q_2) \in \{(0, \bot)\} \times \{(0, \bot), (1, \top)\}$, which are the following input states:

   - $((0, \bot), (0, \bot))$,

- $((0, \bot), (1, \top))$.

And the representation nondeterministic transitions with these two input states are:

- $ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), ((0, \bot), (0, \bot))) = \{(0, \bot), (1, \top)\}$,
- $ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), ((0, \bot), (0, \bot))) = \{(0, \bot)\}$,
- $ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), ((0, \bot), (1, \top))) = \{(1, \top)\}$,
- $ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), ((0, \bot), (1, \top))) = \{(1, \top)\}$.

Since only the representation input symbol and the output state are important to compute the new representation transitions for the input states $(\{(0, \bot)\}, \{(0, \bot), (1, \top)\})$, then we can leave out from these representation transitions unnecessary information and take only their representation input symbols and their output states. We can represent each of them as a pair. Hence, we have the following:

- $((\{\texttt{DECL}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\})$,
- $((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0, \bot)\})$,
- $((\{\texttt{DECL}\}, \{\}, \{\}), \{(1, \top)\})$,
- $((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(1, \top)\})$.

2. From the pairs of representation input symbols and output states, we can derive the representation input symbol and the output state of the new representation transition. The idea for deriving the new representation input symbols is to find the *common represented input symbols* among the diverse representation input symbols in the pairs. For this purpose we can use the diamond operator. Given two pairs of representation input symbols and output states (r_1, P_1) and (r_2, P_2), whenever from r_1 and r_2 some common represented input symbols exist, then for this common represented input symbols we compute the output state by taking the union of P_1 and P_2. This procedure is performed for every two different pairs repeatedly until a fixpoint is reached, i.e. we have a pair of final representation input symbols and their corresponding output states which are complete and non-overlapping.

   For our running example, applying this procedure yields the following pairs of new representation input symbols and output states:

   - $((\{\texttt{DECL}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\})$,
   - $((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\})$,

   Remark 3. Given a set of representation nondeterministic transitions which is complete and non-overlapping with respect to our assumption, we can claim that the application of the above procedure repeatedly (until a fixpoint is reached) will give a complete and non-overlapping representation deterministic transitions. The reason is that in the given representation transitions, which are complete and non-overlapping (with respect to our assumption), for every configuration of input states, all input symbols are completely and non-overlappingly represented. And, to produce the representation input symbol and the output state of a new representation deterministic transition (for a given configuration of input states), we consider every possibility of obtaining the common represented input symbols. This is done by applying the procedure to *every* two different pairs of representation input symbols and output states (of the representation nondeterministic transition). The procedure is performed repeatedly until a fixpoint is reached, which means that no more common represented input symbols can be extracted from them and hence the representation input symbols are non-overlapping one another. $\square$

3. From the given input states $(P_1, \ldots, P_k)$ and every pair of the new representation input symbol and the output state $((E, E_1, E_2), P)$, we can construct the new representation deterministic transitions:

$$dt_k((E, E_1, E_2), (P_1, \ldots, P_k)) = P.$$

For our running example, we have the following new representation transitions:

- from the pair $((\{\texttt{DECL}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\})$, we have:

$$dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0, \bot)\}, \{(0, \bot), (1, \top)\})) = \{(0, \bot), (1, \top)\},$$

- from the pair $((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\})$, we have:

$$dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0, \bot)\}, \{(0, \bot), (1, \top)\})) = \{(0, \bot), (1, \top)\}.$$

With respect to our assumption, we claim that the new representation deterministic transitions produced by our power set construction indeed represents the transitions of the deterministic automaton which are derived from the transitions of the nondeterministic automaton by the power set construction described in section 5.3, since:

- in general, to produce a representation deterministic transition we use the same approach as in the original power set construction, except that we consider only reachable states, but the essence of the construction is still the same.
- in Remark 3, we have argued that given a set of representation nondeterministic transitions which is complete and non-overlapping with respect to our assumption, by our construction we shall also obtain a complete and non-overlapping representation deterministic transitions.

Based on this idea, we define now the reachability construction for power set formally.

To compute the representation transitions $dtrans(\varphi)$ of the deterministic automaton for the formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, where $\varphi = (\exists x : \widetilde{\eta}.\ \psi)$ or $\varphi = (\exists X : \widetilde{\eta}.\ \psi)$, we assume that the set of representation transitions $ndtrans(\varphi)$ of the nondeterministic automaton for the formula φ is given.

First, we define a function that given a k-tuple of (input) states $(P_1, \ldots, P_k) \in St(\varphi)^k$ with $k \in Rank_{\Sigma,+}$, it collects all pairs of representation input symbols and output states from the representation transitions of $ndtrans(\varphi)$ with the input states $(q_1, \ldots, q_k) \in P_1 \times \ldots \times P_k$. We define the function $pstr_k : St(\varphi)^k \to \mathcal{P}(Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi))$ as follows:

$$\begin{array}{rll} & pstr_k\ (P_1, \ldots, P_k) & \\ = \{ & ((E, E_1, E_2), P) \mid & \\ & (ndt_k((E, E_1, E_2), (q_1, \ldots, q_k)) = P) \in ndtrans(\varphi), & \\ & (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2), & \\ & (q_1, \ldots, q_k) \in P_1 \times \ldots \times P_k,\ P \in St(\varphi) & \} \end{array}$$

With this collection of pairs, we can compute the representation input symbol and the output state for the new representation transition for a given k-tuple of input states $(P_1, \ldots, P_k) \in St(\varphi)^k$. To achieve this goal, we proceed step by step. We start by defining a function to compute a new pair of representation input symbol and output state from two pairs of representation input symbols and output states as the function $np : (Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi)) \times (Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi)) \to \mathcal{P}(Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St_\varphi)$:

$np\ (((E', E'_1, E'_2), P_1), ((E'', E''_1, E''_2), P_2)) =$

- <u>if</u> $(E', E'_1, E'_2) \diamond (E'', E''_1, E''_2)$ is defined,
  <u>then</u> $\{((E', E'_1, E'_2) \diamond (E'', E''_1, E''_2), P_1 \cup P_2)\}$,
- <u>otherwise</u>, $\emptyset$.

Let (r, P) be a pair (of representation input symbol and output state) and Zp be a set of pairs (of representation input symbol and output state), we define a function that apply np to the pair (r, P) and every element of Zp. We define the function $sq_{ps} : (Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi)) \times \mathcal{P}(Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi)) \to \mathcal{P}(Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi))$ as follows:

Let $Nzp = \bigcup_{(r', P') \in Zp} np\ ((r, P), (r', P'))$, then:

$sq_{ps}\ ((r, P), Zp) =$

- <u>if</u> $Zp = \emptyset$, <u>then</u> $\emptyset$,
- <u>if</u> $Zp \neq \emptyset$ and $Nzp \neq \emptyset$, <u>then</u> Nzp,
- <u>otherwise</u>, $\{(r, P)\}$.

Now we can use the function sq_{ps} to compute the final collection of pairs of representation input symbols and output states for the representation transition with the input states $(P_1, \ldots, P_k)$ by considering all pairs we have from $pstr_k\ (P_1, \ldots, P_k)$. Let $Zt = pstr_k(P_1, \ldots, P_k)$ and

$$Azp = \bigcup_{(r,P) \in Zt} sq_{ps}\ ((r, P), Zt \setminus \{(r, P)\}),$$

we define function $squeeze_{ps} : \mathcal{P}(Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi)) \rightarrow \mathcal{P}(Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \times St(\varphi))$ as follows:

$squeeze_{ps}\ Zt =$

- <u>if</u> $card(Zt) = 0$, <u>then</u> $\emptyset$,
- <u>if</u> $card(Zt) = 1$, <u>then</u> Zt,
- <u>if</u> $card(Zt) \geq 2$ and $Azp \neq Zt$, <u>then</u> $(squeeze_{ps}\ Azp)$,
- <u>otherwise</u>, Zt.

Having the definitions of $pstr_k$, np, sq_{ps}, and $squeeze_{ps}$, we are ready now to compute the new representation deterministic transitions. For a given rank $k \in Rank_{\Sigma,+}$, we define the function $pstrans_k : \mathcal{P}(St(\varphi))^k \rightarrow \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2))$ that given k sets of reachable states, it computes the new representation deterministic transitions of k-tuples of input states, where each k-tuple of input states comes from the given k sets of reachable states:

$$\begin{array}{ll} & pstrans_k\ (S_1, \ldots, S_k) \\ = \{ & dt_k((E, E_1, E_2), (P_1, \ldots, P_k)) = P \mid \\ & P_1 \in S_1, \ldots, P_k \in S_k, \\ & ((E, E_1, E_2), P) \in squeeze_{ps}\ (pstr_k\ (P_1, \ldots, P_k)) \quad \} \end{array}$$

We define the following function $reach_{ps} : \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St(\varphi)) \times \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St(\varphi)) \rightarrow \mathcal{P}(Dtr(\Sigma, \mathcal{V}_1, \mathcal{V}_2)) \times \mathcal{P}(St(\varphi))$ as the reachability construction for power set:

$reach_{ps}\ (T_{ps}, S_{ps}, \Delta T_{ps}, \Delta S_{ps}) =$

- <u>if</u> $\Delta S_{ps} = \emptyset$,
  <u>then</u>: $(T_{ps} \cup \Delta T_{ps}, S_{ps})$
- <u>otherwise</u>: $reach_{ps}\ (T'_{ps}, S'_{ps}, \Delta T_{ps_{>0}}, \Delta S_{ps_{>0}})$,

where:

- $T'_{ps} = T_{ps} \cup \Delta T_{ps}$
  Note: T'_{ps} collects all representation transitions computed so far.
- $S'_{ps} = S_{ps} \cup \Delta S_{ps}$
  Note: S'_{ps} collects all states which are reachable so far.
- $$\Delta T_{ps_{>0}} = \bigcup_{k \in Rank_{\Sigma,+}} \left(\begin{array}{l} pstrans_k(\underbrace{S_{ps}, \ldots, S_{ps}}_{k-1}, \Delta S_{ps}) \cup \\ pstrans_k(\underbrace{S_{ps}, \ldots, S_{ps}}_{k-2}, \Delta S_{ps}, S'_{ps}) \cup \ldots \cup \\ pstrans_k(\Delta S_{ps}, \underbrace{S'_{ps}, \ldots, S'_{ps}}_{k-1}) \end{array} \right)$$
  Note: $\Delta T_{ps_{>0}}$ computes for every rank $k \in Rank_{\Sigma,+}$ the new representation deterministic transitions, which is performed by the function

pstrans$_k$. As in the cross product construction, by "semi-naive" method, we avoid to compute for every rank $k \in Rank_{\Sigma,+}$, $pstrans_k(\underbrace{S_{ps}, \ldots, S_{ps}}_{k})$, since this means we compute the representation transitions considering only the old reachable states, but this in fact has been computed in the previous recursive call.

- $$\begin{array}{lll} \Delta S_{ps_{>0}} & =\{ & trf_{\varphi}(P) \in St(\varphi) \mid \\ & & (dt_k((E, E_1, E_2), (P_1, \ldots, P_k)) = P) \in \Delta T_{ps_{>0}}, \\ & & k > 0,\ (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2), \\ & & P_1, \ldots, P_k \in S'_{ps} \qquad\qquad\qquad \} \setminus S'_{ps} \end{array}$$

  Note: $\Delta S_{ps_{>0}}$ collects the new reachable states, with function trf_{φ} defined in remark 1.

We can compute now the set of representation transitions $dtrans(\varphi)$ of the deterministic automaton for the formula $(\varphi) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ by using the reachability construction for power set.

Let $(trans_{ps}, sts_{ps}) = reach_{ps}\ (\emptyset, \emptyset, \Delta T_{ps_{=0}}, \Delta S_{ps_{=0}})$, where:

$$\begin{array}{lll} \Delta T_{ps_{=0}} & = \{ & dt_0((E, E_1, E_2), (\,)) = P \mid \\ & & (ndt_0((E, E_1, E_2), (\,)) = P) \in ndtrans(\varphi), \\ & & (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2),\ P \in St(\varphi) \qquad \} \end{array}$$

$$\begin{array}{lll} \Delta S_{ps_{=0}} & = \{ & trf_{\varphi}(P) \in St(\varphi) \mid \\ & & (dt_0((E, E_1, E_2), (\,)) = P) \in \Delta T_{ps_{=0}}, \\ & & (E, E_1, E_2) \in Ext(\Sigma, \mathcal{V}_1, \mathcal{V}_2) \qquad\qquad \} \end{array}$$

Then, we have:

$$dtrans(\varphi) = trans_{ps}.$$

Additionally, sts_{ps} is the set of all reachable states of the deterministic automaton for the formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$.

Example 15. Consider again our running example:

$$\varphi = \exists x : \{\texttt{Decl}\}.\ label_{\texttt{DECL}}(x) \in MSO^*(\Sigma, \mathcal{V}_1 \setminus \{x^{\{\texttt{Decl}\}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

Let us recall the representation nondeterministic transitions $ndtrans(\varphi)$, which was given in Example 13:

$$\begin{array}{lll} ndtrans(\varphi) & = & \{\ ndt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = \{(0, \bot)\}, \\ & & \quad ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), ((0, \bot))) = \{(0, \bot)\}, \\ & & \quad ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), ((1, \top))) = \{(1, \top)\}, \\ & & \quad ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), ((0, \bot), (0, \bot))) = \{(0, \bot), (1, \top)\}, \\ & & \quad ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), ((0, \bot), (0, \bot))) = \{(0, \bot)\}, \\ & & \quad ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), ((1, \top), (0, \bot))) = \{(1, \top)\}, \\ & & \quad ndt_2((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), ((0, \bot), (1, \top))) = \{(1, \top)\} \quad \} \end{array}$$

We compute the representation deterministic transitions $dtrans(\varphi)$ from the representation nondeterministic transitions $ndtrans(\varphi)$ by using the reachability construction of power set. We start by computing:

$$reach_{ps}\ (\emptyset, \emptyset, \Delta T_{ps_{=0}}, \Delta S_{ps_{=0}}),$$

where:

$$\begin{array}{rcl}
\Delta T_{ps_{=0}} & = & \{\ dt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = \{(0, \bot)\}\ \} \\
\Delta S_{ps_{=0}} & = & \{\ \{(0, \bot)\}\ \}.
\end{array}$$

Following our formal definition of $reach_{ps}$, we perform the following recursive call (we give here an additional index to the arguments, just to distinguish the arguments of one recursive call with the other recursive call):

$$reach_{ps}\ (T'_{ps_{(1)}}, S'_{ps_{(1)}}, \Delta T_{ps_{>0(1)}}, \Delta S_{ps_{>0(1)}}),$$

where:

$$\begin{array}{rcl}
T'_{ps_{(1)}} & = & \Delta T_{ps_{=0}} \\
S'_{ps_{(1)}} & = & \Delta S_{ps_{=0}} \\
 & = & \{\ \{(0, \bot)\}\ \} \\
\Delta T_{ps_{>0(1)}} & = & pstrans_1(\Delta S_{ps_{=0}}) \cup \\
 & & pstrans_2(\emptyset, \Delta S_{ps_{=0}}) \cup pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}}) \\
 & = & pstrans_1(\Delta S_{ps_{=0}}) \cup \emptyset \cup pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}}) \\
 & = & pstrans_1(\Delta S_{ps_{=0}}) \cup pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}})^{(\star)}
\end{array}$$

We compute first $pstrans_1(\Delta S_{ps_{=0}})$ and then $pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}})$.

To compute $pstrans_1(\Delta S_{ps_{=0}})$, we need to compute $pstr_1(\{(0, \bot)\})$ and $squeeze_{ps}\ (pstr_1(\{(0, \bot)\}))$:

$$pstr_1(\{(0, \bot)\}) = \{((\{\texttt{DSINGLE}\}, \{\}, \{\}), \{(0, \bot)\})\},$$

and

$$squeeze_{ps}\ (pstr_1(\{(0, \bot)\})) = \{((\{\texttt{DSINGLE}\}, \{\}, \{\}), \{(0, \bot)\})\}.$$

Hence, we have:
$pstrans_1(\Delta S_{ps_{=0}}) = \{dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (\{(0, \bot)\})) = \{(0, \bot)\}\}$

To compute $pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}})$, we need to compute the values of $pstr_2(\{(0, \bot)\}, \{(0, \bot)\})$ and $squeeze_{ps}\ (pstr_2(\{(0, \bot)\}, \{(0, \bot)\}))$:

$$\begin{array}{rcl}
pstr_2(\{(0, \bot)\}, \{(0, \bot)\}) & = & \{\ ((\{\texttt{DECL}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\}), \\
 & & ((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0, \bot)\})\ \}
\end{array}$$

For simplicity of our presentation here, let:

- $p_1 = ((\{\texttt{DECL}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\}),$

- $p_2 = ((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0, \bot)\})$.

We compute $squeeze_{ps}$ $(pstr_2(\{(0, \bot)\}, \{(0, \bot)\}))$ as follows:
First, we compute:

$$\begin{aligned} Azp &= sq_{ps}(p_1, \{p_2\}) \cup sq_{ps}(p_2, \{p_1\}) \\ &= \{p_1, p_2\} \\ &= pstr_2(\{(0, \bot)\}, \{(0, \bot)\}) \end{aligned}$$

and since we have $Azp = pstr_2(\{(0, \bot)\}, \{(0, \bot)\})$, then:

$$\begin{array}{ll} & squeeze_{ps}(pstr_2(\{(0, \bot)\}, \{(0, \bot)\})) \\ = \{ & ((\{\texttt{DECL}\}, \{\}, \{\}), \{(0, \bot), (1, \top)\}), \\ & ((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0, \bot)\}) \qquad\qquad \} \end{array}$$

Hence, we have:

$$\begin{array}{ll} & pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}}) \\ = \{ & dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0, \bot)\}, \{(0, \bot)\})) = \{(0, \bot), (1, \top)\}, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0, \bot)\}, \{(0, \bot)\})) = \{(0, \bot)\} \qquad\qquad \} \end{array}$$

Having the value of $pstrans_1(\Delta S_{ps_{=0}})$ and $pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}})$, we can continue the computation of $\Delta T_{ps_{>0(1)}}$ in $(\star)$:

$$\begin{array}{lll} \Delta T_{ps_{>0(1)}} & = & \ldots \\ & = & pstrans_1(\Delta S_{ps_{=0}}) \cup pstrans_2(\Delta S_{ps_{=0}}, S'_{ps_{(1)}}) \\ & = \{ & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (\{(0, \bot)\})) = \{(0, \bot)\}, \\ & & dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0, \bot)\}, \{(0, \bot)\})) \\ & & \qquad\qquad = \{(0, \bot), (1, \top)\}, \\ & & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0, \bot)\}, \{(0, \bot)\})) \\ & & \qquad\qquad = \{(0, \bot)\} \qquad\qquad \} \end{array}$$

$$\begin{array}{lll} \Delta S_{ps_{>0(1)}} & = & \{\ \{(0, \bot)\}, \{(0, \bot), (1, \top)\}\ \} \setminus S'_{ps_{(1)}} \\ & = & \{\ \{(0, \bot), (1, \top)\}\ \} \end{array}$$

Since $\Delta S_{ps_{>0(1)}} \neq \emptyset$, we perform the following recursive call:

$$reach_{ps}\ (T'_{ps_{(2)}}, S'_{ps_{(2)}}, \Delta T_{ps_{>0(2)}}, \Delta S_{ps_{>0(2)}}),$$

where:

$$\begin{array}{lll} T'_{ps_{(2)}} & = & T'_{ps_{(1)}} \cup \Delta T_{ps_{>0(1)}} \\ S'_{ps_{(2)}} & = & S'_{ps_{(1)}} \cup \Delta S_{ps_{>0(1)}} \\ & = & \{\ \{(0, \bot)\}, \{(0, \bot), (1, \top)\}\ \} \\ \Delta T_{ps_{>0(2)}} & = & pstrans_1(\Delta S_{ps_{>0(1)}}) \cup pstrans_2(S'_{ps_{(1)}}, \Delta S_{ps_{>0(1)}}) \cup \\ & & pstrans_2(\Delta S_{ps_{>0(1)}}, S'_{ps_{(2)}})^{(\star\star)} \end{array}$$

We have to compute the following:

- $pstrans_1(\Delta S_{ps_{>0(1)}})$,
- $pstrans_2(S'_{ps_{(1)}}, \Delta S_{ps_{>0(1)}})$,
- $pstrans_2(\Delta S_{ps_{>0(1)}}, S'_{ps_{(2)}})$.

To compute $pstrans_1(\Delta S_{ps_{>0(1)}})$, we need to compute $pstr_1(\{(0,\bot),(1,\top)\})$ and $squeeze_{ps}$ $(pstr_1(\{(0,\bot),(1,\top)\}))$:

$$\begin{array}{lll} pstr_1(\{(0,\bot),(1,\top)\}) & = \{ & ((\{\texttt{DSINGLE}\},\{\},\{\}),\{(0,\bot)\}), \\ & & ((\{\texttt{DSINGLE}\},\{\},\{\}),\{(1,\top)\}) \quad \} \end{array}$$

For simplicity of our presentation here, let:

- $p'_1 = ((\{\texttt{DSINGLE}\},\{\},\{\}),\{(0,\bot)\})$,
- $p'_2 = ((\{\texttt{DSINGLE}\},\{\},\{\}),\{(1,\top)\})$.

We compute $squeeze_{ps}$ $(pstr_1(\{(0,\bot),(1,\top)\}))$ as follows:
First, we compute:

$$\begin{array}{rcl} Azp & = & sq_{ps}(p'_1,\{p'_2\}) \cup sq_{ps}(p'_2,\{p'_1\}) \\ & = & \{((\{\texttt{DSINGLE}\},\{\},\{\}),\{(0,\bot),(1,\top)\})\} \end{array}$$

and since we have $Azp \neq pstr_1(\{(0,\bot),(1,\top)\})$, then we proceed by having the subsequent recursive call of $squeeze_{ps}$:

$$\begin{array}{rcl} squeeze_{ps}(pstr_1(\{(0,\bot),(1,\top)\})) & = & squeeze_{ps} Azp \\ & = & Azp, \quad \text{since } card(Azp) = 1. \end{array}$$

Hence, we have:

$$\begin{array}{ll} & pstrans_1(\Delta S_{ps_{>0(1)}}) \\ = & \{dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),(\{(0,\bot),(1,\top)\})) = \{(0,\bot),(1,\top)\}\} \end{array}$$

To compute $pstrans_2(S'_{ps_{(1)}}, \Delta S_{ps_{>0(1)}})$, we need to compute the following:

$$\begin{array}{ll} & pstr_2(\{(0,\bot)\},\{(0,\bot),(1,\top)\}) \\ = \{ & ((\{\texttt{DECL}\},\{\},\{\}),\{(0,\bot),(1,\top)\}), \\ & ((\{\texttt{DPAIR}\},\{\},\{\}),\{(0,\bot)\}), \\ & ((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),\{(1,\top)\}) \quad \} \end{array}$$

And $squeeze_{ps}$ $(pstr_2(\{(0,\bot)\},\{(0,\bot),(1,\top)\}))$ has the following value (we only show the final result):

$$\begin{array}{ll} & squeeze_{ps}(pstr_2(\{(0,\bot)\},\{(0,\bot),(1,\top)\})) \\ = \{ & ((\{\texttt{DECL}\},\{\},\{\}),\{(0,\bot),(1,\top)\}), \\ & ((\{\texttt{DPAIR}\},\{\},\{\}),\{(0,\bot),(1,\top)\}) \qquad \} \end{array}$$

Hence, we have:

$$\begin{array}{rl} & pstrans_2(S'_{ps_{(1)}}, \Delta S_{ps_{>0(1)}}) \\ = \{ & dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot)\}, \{(0,\bot),(1,\top)\})) \\ & \qquad = \{(0,\bot),(1,\top)\}, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0,\bot)\}, \{(0,\bot),(1,\top)\})) \\ & \qquad = \{(0,\bot),(1,\top)\} \quad \} \end{array}$$

To compute $pstrans_2(\Delta S_{ps_{>0(1)}}, S'_{ps_{(2)}})$, we need to compute:

1. $pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot)\})$ and $squeeze_{ps}\ (pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot)\}))$,

2. $pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\})$ and $squeeze_{ps}\ (pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\}))$,

where:

$$\begin{array}{rl} & pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot)\}) \\ = & pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\}) \\ = \{ & ((\{\texttt{DECL}\}, \{\}, \{\}), \{(0,\bot),(1,\top)\}), \\ & ((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0,\bot)\}), \\ & ((\{\texttt{DECL}, \texttt{DPAIR}\}, \{\}, \{\}), \{(1,\top)\}) \quad \} \end{array}$$

Therefore, $squeeze_{ps}\ (pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot)\}))$ also has the same value as $squeeze_{ps}\ (pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\}))$, which is the following:

$$\begin{array}{rl} & squeeze_{ps}(pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot)\})) \\ = & squeeze_{ps}(pstr_2(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\})) \\ = \{ & ((\{\texttt{DECL}\}, \{\}, \{\}), \{(0,\bot),(1,\top)\}), \\ & ((\{\texttt{DPAIR}\}, \{\}, \{\}), \{(0,\bot),(1,\top)\}) \quad \} \end{array}$$

Hence, we have:

$$\begin{array}{rl} & pstrans_2(\Delta S_{ps_{>0(1)}}, S'_{ps_{(2)}}) \\ = \{ & dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot)\})) \\ & \qquad = \{(0,\bot),(1,\top)\}, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot)\})) \\ & \qquad = \{(0,\bot),(1,\top)\}, \\ & dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot),(1,\top)\})) \\ & \qquad = \{(0,\bot),(1,\top)\}, \\ & dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot),(1,\top)\})) \\ & \qquad = \{(0,\bot),(1,\top)\} \quad \} \end{array}$$

By using the values of $pstrans_1(\Delta S_{ps_{>0(1)}})$, $pstrans_2(S'_{ps_{(1)}}, \Delta S_{ps_{>0(1)}})$, and $pstrans_2(\Delta S_{ps_{>0(1)}}, S'_{ps_{(2)}})$, we can now continue the computation of $\Delta T_{ps_{>0(2)}}$ in ($\star\star$):

$$
\begin{aligned}
& \Delta T_{ps_{>0(2)}} \\
= \quad & pstrans_1(\Delta S_{ps_{>0(1)}}) \cup pstrans_2(S'_{ps_{(1)}}, \Delta S_{ps_{>0(1)}}) \cup \\
& pstrans_2(\Delta S_{ps_{>0(1)}}, S'_{ps_{(2)}}) \\
= \{ \quad & dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot)\}, \{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0,\bot)\}, \{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\} \qquad \}
\end{aligned}
$$

$$
\begin{aligned}
& \Delta S_{ps_{>0(2)}} \\
= \quad & \{\ \{(0,\bot),(1,\top)\}\ \} \setminus S'_{ps_{(2)}} \\
= \quad & \emptyset
\end{aligned}
$$

Since now $\Delta S_{ps_{>0(2)}} = \emptyset$, we have then the following:

$$
reach_{ps}\ (T'_{ps_{(2)}}, S'_{ps_{(2)}}, \Delta T_{ps_{>0(2)}}, \Delta S_{ps_{>0(2)}}) = (T'_{ps_{(2)}} \cup \Delta T_{ps_{>0(2)}}, S'_{ps_{(2)}})
$$

and the representation transitions $dtrans(\varphi)$ of the automaton for the formula $\varphi = \exists x : \{\texttt{Decl}\}.\ label_{\texttt{DECL}}(x)$ is the following:

$$
\begin{aligned}
& dtrans(\varphi) \\
= \quad & T'_{ps_{(2)}} \cup \Delta T_{ps_{>0(2)}} \\
= \{ \quad & dt_0((\{\texttt{IDENT}, \texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) \\
& \qquad = \{(0,\bot)\}, \\
& dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (\{(0,\bot)\})) \\
& \qquad = \{(0,\bot)\}, \\
& dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot)\}, \{(0,\bot)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot)\}, \{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot)\})) \\
& \qquad = \{(0,\bot),(1,\top)\}, \\
& dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{(0,\bot),(1,\top)\}, \{(0,\bot),(1,\top)\})) \\
& \qquad = \{(0,\bot),(1,\top)\},
\end{aligned}
$$

$$\begin{aligned}
&dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot)\},\{(0,\bot)\}))\\
&\qquad = \{(0,\bot)\},\\
&dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot)\},\{(0,\bot),(1,\top)\}))\\
&\qquad = \{(0,\bot),(1,\top)\},\\
&dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot)\}))\\
&\qquad = \{(0,\bot),(1,\top)\},\\
&dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\}))\\
&\qquad = \{(0,\bot),(1,\top)\} \qquad \}
\end{aligned}$$

Additionally, $S'_{ps_{(2)}} = \{\{(0,\bot)\},\{(0,\bot),(1,\top)\}\}$ is the set of all reachable states of the deterministic automaton for the formula $\varphi = \exists x : \{\texttt{Decl}\}.\ label_{\texttt{DECL}}(x)$.

If we attempt to apply the compression procedure on $ndtrans(\varphi)$, it turns out that the following pairs of representation transitions can be compressed:

- $dt_2((\{\texttt{DECL}\},\{\},\{\}),(\{(0,\bot)\},\{(0,\bot),(1,\top)\})) = \{(0,\bot),(1,\top)\}$,
  $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot)\},\{(0,\bot),(1,\top)\})) = \{(0,\bot),(1,\top)\}$.
- $dt_2((\{\texttt{DECL}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot)\})) = \{(0,\bot),(1,\top)\}$,
  $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot)\})) = \{(0,\bot),(1,\top)\}$.
- $dt_2((\{\texttt{DECL}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\})) = \{(0,\bot),(1,\top)\}$,
  $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\})) = \{(0,\bot),(1,\top)\}$

where the representation input symbol of every compressed representation transition from these three pairs is computed as the following:

$$compress\ ((\{\texttt{DECL}\},\{\},\{\}),(\{\texttt{DPAIR}\},\{\},\{\})) = (\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}).$$

Hence, after compressing the three pairs of representation transitions we have the following (which is also the final result, since no more representation transitions can be compressed):

$$\begin{aligned}
& dtrans(\varphi)\\
= \quad & T'_{ps_{(2)}} \cup \Delta T_{ps_{>0(2)}}\\
= \{ \quad & dt_0((\{\texttt{IDENT},\texttt{INTTYP},\texttt{BOOLTYP}\},\{\},\{\}),(\ ))\\
&\qquad = \{(0,\bot)\},\\
& dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),(\{(0,\bot)\}))\\
&\qquad = \{(0,\bot)\},\\
& dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),(\{(0,\bot),(1,\top)\}))\\
&\qquad = \{(0,\bot),(1,\top)\},\\
& dt_2((\{\texttt{DECL}\},\{\},\{\}),(\{(0,\bot)\},\{(0,\bot)\}))\\
&\qquad = \{(0,\bot),(1,\top)\},\\
& dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot)\},\{(0,\bot)\}))\\
&\qquad = \{(0,\bot)\},\\
& dt_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot)\},\{(0,\bot),(1,\top)\}))\\
&\qquad = \{(0,\bot),(1,\top)\},\\
& dt_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot)\}))\\
&\qquad = \{(0,\bot),(1,\top)\},\\
& dt_2((\{\texttt{DECL},\texttt{DPAIR}\},\{\},\{\}),(\{(0,\bot),(1,\top)\},\{(0,\bot),(1,\top)\}))\\
&\qquad = \{(0,\bot),(1,\top)\},
\end{aligned}$$

□

We give a comparison of the number of states and transitions using our representation and those which are constructed by following the formal construction given in section 5.3. This is shown in table 7.12.

	Non-represented	Represented
Number of States	16	2
Number of Transition	540	8

Table 7.12: A comparison of the number of states and transitions (represented and non-represented) for the deterministic automaton of the formula $\exists x : \{\texttt{Decl}\}.\ label_{\texttt{DECL}}(x)$

We can have a significant reduction of the number of states, since the number of states of the nondeterministic automaton itself (by our projection construction) has been already reduced, from four states to two states (cf. table 7.10). If all reachable states from the nondeterministic automaton are considered, we would then have the following states in the deterministic automaton: $\{\}$, $\{(0,\bot)\}$, $\{(1,\top)\}$, and $\{(0,\bot),(1,\top)\}$. But, by considering only the reachable states in the power set construction, we can have less number of states, where only the states $\{(0,\bot)\}$ and $\{(0,\bot),(1,\top)\}$ are reachable.

Those 540 transitions in table 7.12 are obtained by following the formal construction given in section 5.3, where every input symbol and every state as the input states of the transitions are considered:

- 8 transitions having the input symbols with the first component: $\texttt{IDENT}^{(\varepsilon,\texttt{Ident})}$, where we consider 2 node variables ($y^{\{\texttt{Ident}\}}$ and $z^{\{\texttt{Ident}\}}$), 1 node set variable ($X^{\{\texttt{Ident}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{INTTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 2 transitions having the input symbols with the first component: $\texttt{BOOLTYP}^{(\varepsilon,\texttt{TypExp})}$, where we consider no node variables, 1 node set variable ($Y^{\{\texttt{TypExp}\}}$), and no input state,
- 16 transitions having the input symbols with the first component: $\texttt{DSINGLE}^{(\texttt{Decl},\texttt{DList})}$, where we consider no node variables, no node set variable, and 16 combinations of input states,
- 256 transitions having the input symbols with the first component: $\texttt{DPAIR}^{(\texttt{Decl DList},\texttt{DList})}$, where we consider no node variable, no node set variable, and 256 combinations of input states,
- 256 transitions having the input symbols with the first component: $\texttt{DECL}^{(\texttt{Ident TypExp},\texttt{Decl})}$, where we consider no node variable (since x is the quantified variable), no node set variable, and 256 combinations of input states.

The approach of the power set construction we discussed so far does not directly work if we aim to produce the representation deterministic transitions of a quantification formula, where the subformula contains an attribute formula and we quantify over a node variable that occurs in the attribute formula. In the following, we show the problem and then propose a refinement to our previous approach such that it also works for this case.

What if an attribute formula is involved?

We use an example to illustrate the problem that happens when an attribute formula is involved in a quantification formula, where we quantify over a node variable that occurs in the attribute formula. Consider the following formula:

$$\varphi = (\exists x : \{\texttt{Decl}\}.\ gt(\langle often, x\rangle(\langle name, y\rangle), null)) \in MSO^*(\Sigma, \mathcal{V}_1 \setminus \{x^{\{\texttt{Decl}\}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

Recall the set of representation transitions for the subformula:

$$\psi = (gt(\langle num, x\rangle(\langle name, y\rangle), null)) \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}},$$

which was given in the example 9. To construct the set of representation nondeterministic transitions for the formula φ, we apply the projection construction to the set of representation transitions for the formula ψ, where we have the following:

$$\begin{aligned}
& ndtrans(\varphi) \\
= \{\quad & ndt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) \\
&\qquad = \{((d_x, d_y), \bot)\}, \\
& ndt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) \\
&\qquad = \{((d_x, \varepsilon_y), \bot)\}, \\
& ndt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) \\
&\qquad = \{((d_x, d_y), \bot)\}, \\
& ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (((d_x, d_y), \bot))) \\
&\qquad = \{((d_x, d_y), \bot)\}, \\
& ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (((d_x, nd_y), \bot))) \\
&\qquad = \{((d_x, 1.nd_y), \bot)\}, \\
& ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (((nd_x, d_y), \top))) \\
&\qquad = \{((1.nd_x, d_y), \top)\}, \\
& ndt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (((nd_x, nd_y), \top))) \\
&\qquad = \{((1.nd_x, 1.nd_y), \top)\}, \\
& ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), (((d_x, d_y), \bot), ((d_x, d_y), \bot))) \\
&\qquad = \{((d_x, d_y), \bot), ((\varepsilon_x, d_y), \top)\}, \\
& ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), (((d_x, d_y), \bot), ((d_x, nd_y), \bot))) \\
&\qquad = \{((d_x, 2.nd_y), \bot), ((\varepsilon_x, 2.nd_y), \top)\}, \\
& ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), (((d_x, d_y), \bot), ((nd_x, d_y), \top))) \\
&\qquad = \{((2.nd_x, d_y), \top)\}, \\
& ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), (((d_x, d_y), \bot), ((nd_x, nd_y), \top))) \\
&\qquad = \{((2.nd_x, 2.nd_y), \top)\}, \\
& ndt_2((\{\texttt{DECL}\}, \{\}, \{\}), (((d_x, nd_y), \bot), ((d_x, d_y), \bot))) \\
&\qquad = \{((d_x, 1.nd_y), \bot), ((\varepsilon_x, 1.nd_y), \top)\},
\end{aligned}$$

$$
\begin{aligned}
&ndt_2((\{\texttt{DECL}\},\{\},\{\}),(((d_x,nd_y),\bot),((nd_x,d_y),\top))) \\
&\qquad\qquad\qquad\qquad = \{((2.nd_x,1.nd_y),\top)\}, \\
&ndt_2((\{\texttt{DECL}\},\{\},\{\}),(((nd_x,d_y),\top),((d_x,d_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((1.nd_x,d_y),\top)\}, \\
&ndt_2((\{\texttt{DECL}\},\{\},\{\}),(((nd_x,d_y),\top),((d_x,nd_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((1.nd_x,2.nd_y),\top)\}, \\
&ndt_2((\{\texttt{DECL}\},\{\},\{\}),(((nd_x,nd_y),\top),((d_x,d_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((1.nd_x,1.nd_y),\top)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,d_y),\bot),((d_x,d_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((d_x,d_y),\bot)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,d_y),\bot),((d_x,nd_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((d_x,2.nd_y),\bot)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,d_y),\bot),((nd_x,d_y),\top))) \\
&\qquad\qquad\qquad\qquad = \{((2.nd_x,d_y),\top)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,d_y),\bot),((nd_x,nd_y),\top))) \\
&\qquad\qquad\qquad\qquad = \{((2.nd_x,2.nd_y),\top)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,nd_y),\bot),((d_x,d_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((d_x,1.nd_y),\bot)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,nd_y),\bot),((nd_x,d_y),\top))) \\
&\qquad\qquad\qquad\qquad = \{((2.nd_x,1.nd_y),\top)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((nd_x,d_y),\top),((d_x,d_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((1.nd_x,d_y),\top)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((nd_x,d_y),\top),((d_x,nd_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((1.nd_x,2.nd_y),\top)\}, \\
&ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((nd_x,nd_y),\top),((d_x,d_y),\bot))) \\
&\qquad\qquad\qquad\qquad = \{((1.nd_x,1.nd_y),\top)\} \qquad \}
\end{aligned}
$$

We apply now the power set construction in order to obtain the set of representation deterministic transitions from the nondeterministic one. We start by constructing the representation transitions for the nullary ranked alphabet, where we have the following representation transitions:

- $dt_0((\{\texttt{INTTYP},\texttt{BOOLTYP}\},\{\},\{\}),(\,)) = \{((d_x,d_y),\bot)\}$,
- $dt_0((\{\texttt{IDENT}\},\{y\},\{\}),(\,)) = \{((d_x,\varepsilon_y),\bot)\}$,
- $dt_0((\{\texttt{IDENT}\},\{\overline{y}\},\{\}),(\,)) = \{((d_x,d_y),\bot)\}$.

Up to this point, the reachable states are the following:

- $\{((d_x,d_y),\bot)\}$,
- $\{((d_x,nd_y),\bot)\}$.
  Note that instead of $\{((d_x,\varepsilon_y),\bot)\}$, we have $\{((d_x,nd_y),\bot)\}$ as a reachable state, since by our construction a particular symbolic representation of Dewey notation that is contained in a reachable state, in this example ε_y, should be transformed into a general one via the function trf_φ (cf. the definition of $\Delta S_{ps=0}$ in page 111).

We proceed further to construct the representation transitions with these two reachable states as input states and by our power set construction, we have the following:

1. $dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),(\{((d_x,d_y),\bot)\}))$
   $= \{((d_x,d_y),\bot)\}$,
2. $dt_1((\{\texttt{DSINGLE}\},\{\},\{\}),(\{((d_x,nd_y),\bot)\})$
   $= \{((d_x,1.nd_y),\bot)\}$,
3. $dt_2((\{\texttt{DECL}\},\{\},\{\}),(\{((d_x,d_y),\bot)\},\{((d_x,d_y),\bot)\}))$
   $= \{((d_x,d_y),\bot),((\varepsilon_x,d_y),\top)\}$,
4. $dt_2((\{\texttt{DECL}\},\{\},\{\}),(\{((d_x,d_y),\bot)\},\{((d_x,nd_y),\bot)\}))$
   $= \{((d_x,2.nd_y),\bot),((\varepsilon_x,2.nd_y),\top)\}$,
5. $dt_2((\{\texttt{DECL}\},\{\},\{\}),(\{((d_x,nd_y),\bot)\},\{((d_x,d_y),\bot)\}))$
   $= \{((d_x,1.nd_y),\bot),((\varepsilon_x,1.nd_y),\top)\}$,
6. $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{((d_x,d_y),\bot)\},\{((d_x,d_y),\bot)\}))$
   $= \{((d_x,d_y),\bot)\}$,
7. $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{((d_x,d_y),\bot)\},\{((d_x,nd_y),\bot)\}))$
   $= \{((d_x,2.nd_y),\bot)\}$,
8. $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{((d_x,nd_y),\bot)\},\{((d_x,d_y),\bot)\}))$
   $= \{((d_x,1.nd_y),\bot)\}$.

And the new reachable states are the following (in addition to the two previous reachable states):

- $\{((d_x,d_y),\bot),((nd_x,d_y),\top)\}$ (from the representation transition 3),
- $\{((d_x,nd_y),\bot),((nd_x,nd_y),\top)\}$ (from the representation transition 4).

We proceed further and to exhibit the problem, suppose we are at the point to compute the representation transitions for the input symbol $(\{\texttt{DPAIR}\},\{\},\{\})$ with the input states:

$$(\{((d_x,d_y),\bot),((nd_x,d_y),\top)\},\{((d_x,d_y),\bot),((nd_x,d_y),\top)\}).$$

By our power set construction we have then the following:

$$\begin{aligned}&dt_2((\{\texttt{DPAIR}\},\{\},\{\}),\\&\qquad(\{((d_x,d_y),\bot),((nd_x,d_y),\top)\},\{((d_x,d_y),\bot),((nd_x,d_y),\top)\}))\\&\qquad\qquad= \{((d_x,d_y),\bot),((1.nd_x,d_y),\top),((2.nd_x,d_y),\top)\}.^{(\star)}\end{aligned}$$

Note that:

(a) the state $((d_x,d_y),\bot)$ in the output state is obtained from the following representation transition:

$$ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,d_y),\bot),((d_x,d_y),\bot))) = \{((d_x,d_y),\bot)\}$$

(b) the state $((1.nd_x,d_y),\top)$ in the output state is obtained from the following representation transition:

$$ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((nd_x,d_y),\top),((d_x,d_y),\bot))) = \{((1.nd_x,d_y),\top)\}$$

(c) the state $((2.nd_x, d_y), \top)$ in the output state is obtained from the following representation transition:

$$ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (((d_x, d_y), \bot), ((nd_x, d_y), \top))) = \{((2.nd_x, d_y), \top)\}$$

It turns out that the symbolic representation nd_x in (b) does not necessarily represent the same Dewey notation as represented by nd_x in (c). The reason is that x is guessed at different nodes (some nodes are guessed from the left branch of the node labeled by `DPAIR` and some others are guessed from its right branch). But, in the representation transition $(\star)$, which is derived by the power set construction, the symbolic representation nd_x becomes indistinguishable. Hence, there is a need to give an additional index to every symbolic representation nd_x in order to distinguish them. The representation transition in $(\star)$ now becomes the following:

$$\begin{aligned}&dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}),\\ &\qquad(\{((d_x, d_y), \bot), ((nd_x^1, d_y), \top)\}, \{((d_x, d_y), \bot), ((nd_x^2, d_y), \top)\}))\\ &\qquad\qquad= \{((d_x, d_y), \bot), ((1.nd_x^1, d_y), \top), ((2.nd_x^2, d_y), \top)\}.\end{aligned}$$

But, this still leaves some problems. By the definition of the computation of the new reachable state (cf. $\Delta S_{ps_{>0}}$ in page 111), from the last representation transition we shall not obtain a new reachable state, since after applying the transformation function trf_φ to its output state we have the following (we also drop the additional index, since a new index should be newly selected when we later want to have the reachable state as an input state of a representation transition):

$$\begin{aligned}& trf_\varphi(\{((d_x, d_y), \bot), ((1.nd_x, d_y), \top), ((2.nd_x, d_y), \top)\})\\ = \;& \{((d_x, d_y), \bot), ((nd_x, d_y), \top)\}.\end{aligned}$$

From this computation of reachable state, $((1.nd_x, d_y), \top)$ and $((2.nd_x, d_y), \top)$ are now represented as $((nd_x, d_y), \top)$ in the reachable state. But, the symbolic representation $1.nd_x$ in $((1.nd_x, d_y), \top)$ and $2.nd_x$ in $((2.nd_x, d_y), \top)$ represent different nodes which are guessed for the node variable x. Hence, they can not be represented as one symbolic representation nd_x which represents only one node. The idea to overcome this problem is to use the symbolic representation of a set of nodes which was introduced in section 7.2. By using this symbolic representation we are able to represent the union of the guessed nodes, which can be realized by using the symbolic representation $sms_x(s_1, s_2)$, as one symbolic representation.

We summarize two important things which we should consider for refining our power set construction:

- Instead of using a symbolic representation of a node for representing the value of the quantified variable, we need to use the symbolic representation of *a set of nodes* to represent the guessed nodes for the quantified node variable.

- When a representation deterministic transition is constructed, there is a need to have an additional index for every symbolic representation snd_x of the quantified variable x, which occurs in the input states, uniquely. Hence, there is a clear reference for every symbolic representation of the quantified variable, which occurs in the output state, to every symbolic representation of the quantified variable, which occurs in the input states. In our symbolic representation of a set of nodes (cf. section 7.2), we have already included this additional index as a superscript.

Based on this consideration, in the following we give the idea to refine our power set construction informally:

1. We start by computing the representation transitions for the nullary ranked alphabet. These representation transitions can be derived from the representation nondeterministic transitions for the nullary ranked alphabet, where we transform every symbolic representation of a node for the quantified variable into its corresponding symbolic representation of a set of nodes. For the quantified variable x, there could be only two possibilities of symbolic representation in the output state of the representation nondeterministic transition for the nullary ranked alphabet:

   - in case we do not make a guess, then we have d_x and we transform it to $\emptyset_x$,
   - in case we make a guess, then we have ε_x and we transform it to $sc\varepsilon_x$.

   Only the symbolic representation for x are modified, the rest structure of the output state is left unchanged. This transformation can be realized by induction on the structure of the state.

2. The new reachable states are computed in the same way as before, where we transform every particular symbolic representation into a general symbolic representation. This is realized by applying the function trf_φ in page 76 to the output state of every newly derived representation transition. Using the current reachable states, we compute other representation transitions with the combinations of the reachable states as the configurations of the input states. Since we do not keep the additional index (as superscript in our symbolic representation) in a reachable state, then we should give such index when this reachable state becomes an input state. In general, the l-th input state (with $l \in [k]$) of a representation transition for the rank $k \in Rank_\Sigma$ has the following form (suppose x is the quantified variable):

   $$\{(\langle c_0\rangle\emptyset_x\langle c'_0\rangle, \bot), (\langle c_1\rangle snd_x^{(l,1)}\langle c'_1\rangle, \top), \ldots, (\langle c_m\rangle snd_x^{(l,m_l)}\langle c'_m\rangle, \top)\},$$

   where $m \geq 0$, $c_0, c'_0, \ldots, c_m, c'_m$ are the rest structure of a state and the indexing is done as it is shown above. This indexing scheme is done to every symbolic representation of x which occurs in every input state of a representation transition.

3. Having k-tuple of such input states, i.e. $P_1, \ldots, P_k$, we compute the new representation transition by searching the representation nondeterministic transitions with the input states $(q_1, \ldots, q_k) \in P_1 \times \ldots \times P_k$, in particular we search only those representation transitions with the input states $(q_1, \ldots, q_k)$, where there is at most one $j \in [k]$, whose second component is $\top$ (recall that $q_j \in St_{non}(\varphi)$, with $\varphi = \exists x : \widetilde{\eta}.\ \psi$).

   To successfully search from the set of representation nondeterministic transitions, the symbolic representation $snd_x^{(j,m_j)}$ that occurs in one of the q_j (the state where a guess of x is made) has to be converted to the symbolic representation nd_x. Similarly, the symbolic representation $\emptyset_x$ has to be converted to the symbolic representation d_x. The reason is that in the representation nondeterministic transitions we do not recognize the symbolic representation of set of nodes, instead we use the symbolic representation of a node, i.e. nd_x (and d_x, respectively).

   On the other hand, the symbolic representation ε_x (and $j.nd_x$, respectively) that occurs in the output state of a representation nondeterministic transition with the input states $(q'_1, \ldots, q'_k)$, where for every $j \in [k]$, q'_j is derived from q_j by converting $snd_x^{(j,m_j)}$ (if it exists) into nd_x, has to be converted to the symbolic representation $sc\varepsilon_x$ (and $j.snd_x^{(j,m_j)}$, respectively). Similarly, the symbolic representation d_x is also converted into $\emptyset_x$. The reason is that this output state of the representation nondeterministic transition will become an element of the output state of the representation deterministic transition with the input states $(P_1, \ldots, P_k)$, where in $P_1, \ldots, P_k$ a symbolic representations of a set of nodes for the node variable x, are also used.

4. After we successfully find the representation nondeterministic transitions with the input states $(q'_1, \ldots, q'_k)$ and after properly convert the output state, we can extract the representation input symbols and the converted output state as a pair and then perform the function $squeeze_{ps}$ described previously. In our power set construction described earlier (where we do not take into account an attribute formula), at the end of the computation with the function $squeeze_{ps}$ we would get the final representation input symbol and output state for the representation deterministic transition with the input states $(P_1, \ldots, P_k)$.

   In this case, at the end of the computation with the function $squeeze_{ps}$ we only get a final representation input symbol, but the output state has to be processed further to get the final result. The motivation of this final processing of the output state is to merge the elements of the output state containing different symbolic representations of the quantified variable x, but having the same rest structure of element. The result is one element which keeps the common rest structure and only the different symbolic representations are merged. Intuitively, the merging of two different symbolic representations can be seen as the union of two different sets of nodes which are guessed for the node variable x. The merging of two different symbolic representation for the node variable x, say s_1 and s_2, is represented by $sms_x(s_1, s_2)$. The elements of the output state which can not be merged are left unchanged.

We show now how this idea is applied to our previous example, viz. the following formula:

$$\varphi = (\exists x : \{\texttt{Decl}\}.\ gt(\langle often, x\rangle(\langle name, y\rangle), null)) \\ \in MSO^*(\Sigma, \mathcal{V}_1 \setminus \{x^{\{\texttt{Decl}\}}\}, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}.$$

We start by constructing the representation transitions for the nullary ranked alphabet. Following the newly proposed idea, we have the following representation transitions:

- $dt_0((\{\texttt{INTTYP}, \texttt{BOOLTYP}\}, \{\}, \{\}), (\,)) = \{((\emptyset_x, d_y), \bot)\}$,
- $dt_0((\{\texttt{IDENT}\}, \{y\}, \{\}), (\,)) = \{((\emptyset_x, \varepsilon_y), \bot)\}$,
- $dt_0((\{\texttt{IDENT}\}, \{\overline{y}\}, \{\}), (\,)) = \{((\emptyset_x, d_y), \bot)\}$.

Note that since x is the quantified variable, then instead of having d_x in the output states of these three representation transitions, we have $\emptyset_x$. The symbolic representation of non-quantified variable y, i.e. d_y and ε_y remain the same.

Up to this point, the reachable states are the following: (which are computed by applying the function trf_φ to the output state of the previous three representation transitions, cf. definition of $\Delta S_{ps=0}$ in page 111)

- $\{((\emptyset_x, d_y), \bot)\}$,
- $\{((\emptyset_x, nd_y), \bot)\}$.

We proceed further to construct the representation transitions with these two reachable states as input states and by our refined power set construction, we have the following:

1. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (\{((\emptyset_x, d_y), \bot)\}))$
   $= \{((\emptyset_x, d_y), \bot)\}$,
2. $dt_1((\{\texttt{DSINGLE}\}, \{\}, \{\}), (\{((\emptyset_x, nd_y), \bot)\}))$
   $= \{((\emptyset_x, 1.nd_y), \bot)\}$,
3. $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{((\emptyset_x, d_y), \bot)\}, \{((\emptyset_x, d_y), \bot)\}))$
   $= \{((\emptyset_x, d_y), \bot), ((sc\varepsilon_x, d_y), \top)\}$,
4. $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{((\emptyset_x, d_y), \bot)\}, \{((\emptyset_x, nd_y), \bot)\}))$
   $= \{((\emptyset_x, 2.nd_y), \bot), ((sc\varepsilon_x, 2.nd_y), \top)\}$,
5. $dt_2((\{\texttt{DECL}\}, \{\}, \{\}), (\{((\emptyset_x, nd_y), \bot)\}, \{((\emptyset_x, d_y), \bot)\}))$
   $= \{((\emptyset_x, 1.nd_y), \bot), ((sc\varepsilon_x, 1.nd_y), \top)\}$,
6. $dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (\{((\emptyset_x, d_y), \bot)\}, \{((\emptyset_x, d_y), \bot)\}))$
   $= \{((\emptyset_x, d_y), \bot)\}$,

7. $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{((\emptyset_x,d_y),\bot)\},\{((\emptyset_x,nd_y),\bot)\}))$
   $=\{((\emptyset_x,2.nd_y),\bot)\}$,
8. $dt_2((\{\texttt{DPAIR}\},\{\},\{\}),(\{((\emptyset_x,nd_y),\bot)\},\{((\emptyset_x,d_y),\bot)\}))$
   $=\{((\emptyset_x,1.nd_y),\bot)\}$.

And the new reachable states are the following (in addition to the two previous reachable states):

- $\{((\emptyset_x,d_y),\bot),((snd_x,d_y),\top)\}$ (from the representation transition 3).

- $\{((\emptyset_x,nd_y),\bot),((snd_x,nd_y),\top)\}$ (from the representation transition 4).

Note that instead of having a particular symbolic representation $sc\varepsilon_x$ in these reachable states, by the function trf_φ we have the general symbolic representation snd_x.

We proceed further and to exhibit that the previous problem can be solved, suppose we are at the same point where the problem occurs. We compute the representation transitions for the input symbol $(\{\texttt{DPAIR}\},\{\},\{\})$, but now instead of having the input states:

$$(\{((d_x,d_y),\bot),((nd_x,d_y),\top)\},\{((d_x,d_y),\bot),((nd_x,d_y),\top)\}),$$

which led to the problem, we have the following input states:

$$(\{((\emptyset_x,d_y),\bot),((snd_x^{(1,1)},d_y),\top)\},\{((\emptyset_x,d_y),\bot),((snd_x^{(2,1)},d),\top)\}).$$

Note that the additional superscript indices are added by following our indexing scheme. We recall that this additional index is needed since without this additional index, the symbolic representation snd_x that occurs in the first input state is indistinguishable with the symbolic representation snd_x in the second input state, but in fact they do not necessarily represent the same set of guessed nodes for x.

By our refined power set construction we have then the following:

$$\begin{aligned}&dt_2((\{\texttt{DPAIR}\},\{\},\{\}),\\ &\quad(\{((\emptyset_x,d_y),\bot),((snd_x^{(1,1)},d_y),\top)\},\{((\emptyset_x,d_y),\bot),((snd_x^{(2,1)},d),\top)\}))\\ &\qquad\qquad=\{((\emptyset_x,d_y),\bot),((1.snd_x^{(1,1)},d_y),\top),((2.snd_x^{(2,1)},d_y),\top)\}.^{(\star)}\end{aligned}$$

The output state is computed as the following:

(a) the state $((\emptyset_x,d_y),\bot)$ in the output state is obtained from the following representation transition:

$$ndt_2((\{\texttt{DPAIR}\},\{\},\{\}),(((d_x,d_y),\bot),((d_x,d_y),\bot)))=\{((d_x,d_y),\bot)\}$$

Note that we choose the representation nondeterministic transition with the first and the second input state $((d_x,d_y),\bot)$ by considering the elements of the first and the second input state of the representation deterministic transition we want to construct, i.e. $((\emptyset_x,d_y),\bot)$. According to our scheme, to successfully find the required representation nondeterministic transition, the symbolic representation $\emptyset_x$ has to be converted into d_x. On the other hand, the symbolic representation d_x in the output state of the representation nondeterministic transition is converted to $\emptyset_x$.

(b) the state $((1.snd_x^{(1,1)}, d_y), \top)$ in the output state is obtained from the following representation transition:

$$ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (((nd_x, d_y), \top), ((d_x, d_y), \bot))) = \{((1.nd_x, d), \top)\}$$

Note that we choose the representation nondeterministic transition with the first input state $((nd_x, d_y), \top)$ by considering the elements of the first input state of the representation deterministic transition we want to construct, i.e. $((snd_x^{(1,1)}, d_y), \top)$, and the second input state of the representation nondeterministic transition, i.e. $((d_x, d_y), \bot)$ is by considering the elements of the second input state, i.e. $((\emptyset_x, d_y), \bot)$ of the representation deterministic transition. To successfully find the required representation nondeterministic transition, the symbolic representation $snd_x^{(1,1)}$ is converted into nd_x and similarly $\emptyset_x$ is converted to d_x. On the other hand, the symbolic representation $1.nd_x$ in the output state of the representation nondeterministic transition is converted to $1.snd_x^{(1,1)}$.

(c) the state $((2.snd_x, d), \top)$ in the output state is obtained from the following representation transition:

$$ndt_2((\{\texttt{DPAIR}\}, \{\}, \{\}), (((d_x, d_y), \bot), ((nd_x, d_y), \top))) = \{((2.nd_x, d_y), \top)\}$$

Note that we choose the representation nondeterministic transition with the first input state $((d_x, d_y), \bot)$ by considering the elements of the first input state of the representation deterministic transition we want to construct, i.e. $((\emptyset_x, d_y), \bot)$, and the second input state of the representation nondeterministic transition, i.e. $((nd_x, d_y), \top)$ by considering the elements of the second input state, i.e. $((snd_x^{(2,1)}, d), \top)$ of the representation deterministic transition. To successfully find the required representation nondeterministic transition, the symbolic representation $snd_x^{(2,1)}$ is converted into nd_x and similarly $\emptyset_x$ is converted to d_x. On the other hand, the symbolic representation $2.nd_x$ in the output state of the representation nondeterministic transition is converted to $2.snd_x^{(2,1)}$.

The output state of the representation transition ($\star$) has to be processed further, in order to merge some elements of the output state, whenever possible. Among the three elements of the output state, $((1.snd_x^{(1,1)}, d_y), \top)$ and $((2.snd_x^{(2,1)}, d_y), \top)$ can be merged together into one element. They differ only in the symbolic representation of x, where the first one contains $1.snd_x^{(1,1)}$ and the latter contains $2.snd_x^{(2,1)}$. By merging these two elements, we have the following new element of the output state:

$$((sms_x(1.snd_x^{(1,1)}, 2.snd_x^{(2,1)}), d_y), \top).$$

The element $((\emptyset_x, d_y), \bot)$ in the output state of the representation transition $(\star)$ can not be merged to any other element, since its components other than the symbolic representation of x are also differ with the other elements, e.g. in this element we have $\bot$, but both other elements have $\top$.

Hence, instead of representation transition $(\star)$, we have the following representation transition:

$$\begin{aligned} &dt_2((\{\texttt{DPAIR}\}, \{\}, \{\}),\\ &\quad(\{((\emptyset_x, d_y), \bot), ((snd_x^{(1,1)}, d_y), \top)\}, \{((\emptyset_x, d_y), \bot), ((snd_x^{(2,1)}, d), \top)\}))\\ &\qquad\qquad = \{((\emptyset_x, d_y), \bot), ((sms_x(1.snd_x^{(1,1)}, 2.snd_x^{(2,1)}), d_y), \top)\}. \end{aligned}$$

By our computation of reachable state, considering the output state of this representation transition, we would have $\{((\emptyset_x, d_y), \bot), (snd_x, d_y), \top)\}$ as a reachable state (which is no new reachable state, i.e. we already had this state as a reachable state). Note that here we have no more problem of representation as before, since now in the reachable state, instead of having nd_x which represents only one guessed node for x, we have a symbolic representation snd_x which represents a set of guessed nodes for x. This symbolic representation snd_x does not come from two separate symbolic representations of a node (as we had before), instead it comes from one symbolic representation of a set of nodes, i.e. $sms_x(1.snd_x^{(1,1)}, 2.snd_x^{(2,1)})$.

7.4 Defining The Accepting Functions

We define the accepting functions also inductively on the structure of the formula. To define the accepting functions, we do not introduce a special representation for them. Instead, we define them directly from the structure of the formula. This is possible, if we keep also the structure of the state which corresponds to the structure of the formula. Since the accepting function is used after we obtain a state which is reached after the automaton has run on a tree, by applying the definition of the accepting function inductively on the structure of this state, it can be decided eventually whether a tree is accepted or not.

In the following, we shall define the accepting function for every formula $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$, that given a state q (a concrete value of state and *not* a representation of state) and a well-marked tree t, it decides whether q is an accepting state or not. The tree t (on which the automaton runs) is needed in the accepting function, in particular when an attribute formula is involved in φ. By having the tree t, we are able to get nodes of t (which are specified by their path information in the state of the attribute formula) and get the required attribute values from these nodes to be evaluated in the accepting function of the attribute formula.

Let $\varphi \in MSO^*(\Sigma, \mathcal{V}_1, \mathcal{V}_2)_{\mathcal{A},\mathcal{F}}$ and $\mathcal{M}^{(\varphi)}_{dec,int_{\mathcal{F}}} = (Q^{(\varphi)}, \Sigma, \delta^{(\varphi)}, F^{(\varphi)})$ be the deterministic bottom-up tree automaton for φ. Given a family $dec = \{dec_t | t \in TR(\Sigma)\}$ of interpretation functions and the interpretation function $int_{\mathcal{F}}$, we define the function $accept_{\varphi}^{dec,int_{\mathcal{F}}} : Q^{(\varphi)} \times TR(\Sigma) \to Bool$ inductively on the structure of φ as the following (we assume that the family $ids = \{ids_t | t \in TR(\Sigma)\}$ of functions is always given):

- $\varphi = true$
  $accept_{\varphi}^{dec,int_{\mathcal{F}}}(q,t) = True$
- For the following formulas:
  - $\varphi = label_{\sigma}(x)$,
  - $\varphi = x \in X$,
  - $\varphi = (x == y)$,
  - $\varphi = type_{\eta}(x)$,

  we have the following accepting function:
  $accept_{\varphi}^{dec,int_{\mathcal{F}}}(q,t) =$
  - <u>if</u> $q = 1$, <u>then</u>: $True$,
  - <u>otherwise</u>: $False$
- For the following formulas:
  - $\varphi = edge_i(x,y)$,
  - $\varphi = x\ over\ y$,

  we have the following accepting function:
  $accept_{\varphi}^{dec,int_{\mathcal{F}}}(q,t) =$
  - <u>if</u> $q = 2$, <u>then</u>: $True$,
  - <u>otherwise</u>: $False$
- $\varphi \in T^{Bool}_{\mathcal{I}\langle\mathcal{A},\mathcal{V}_1\rangle,\mathcal{F}}$
  Let $Var(\varphi)$ be the set of variables in φ and $l = card(Var(\varphi))$.
  $accept_{\varphi}^{dec,int_{\mathcal{F}}}((v_1,\ldots,v_l),t) = \tau_{\theta}(dec_{peel(t)}, int_{\mathcal{F}})(\varphi)$,
  where:
  for every $j \in [l]$, $\theta(x_j) = ids_{peel(t)}(v_j)$.
- $\varphi = \neg\psi$
  $accept_{\varphi}^{dec,int_{\mathcal{F}}}(q,t) = \neg\ accept_{\psi}^{dec,int_{\mathcal{F}}}(q,t)$
- $\varphi = \psi_1 \wedge \psi_2$
  $accept_{\varphi}^{dec,int_{\mathcal{F}}}((q_1,q_2),t) = accept_{\psi_1}^{dec,int_{\mathcal{F}}}(q_1,t)\ \wedge\ accept_{\psi_2}^{dec,int_{\mathcal{F}}}(q_2,t)$
- $\varphi = \exists x : \{\widetilde{\eta}\}.\ \psi$
  - <u>if</u> ψ does not contain an attribute formula in which the node variable x occurs, <u>then</u>:

$$accept_{\varphi}^{dec,int_{\mathcal{F}}}(q,t) = \bigvee_{(s,\top)\in q} accept_{\psi}^{dec,int_{\mathcal{F}}}(s,t)$$

– <u>if</u> ψ contains an attribute formula in which the node variable x occurs, <u>then</u>:

$$accept_{\varphi}^{dec,int_{\mathcal{F}}}(q,t) = \bigvee_{(s,\top)\in q} \; \bigvee_{e \in get_{x,\psi}(s)} accept_{\psi}^{dec,int_{\mathcal{F}}}(subs_{x,\psi}(s,e),t),$$

where:

* $get_{x,\psi}$ is a function that given a state $s \in Q^{(\psi)}$, it returns the set of guessed nodes (specified by Dewey notations) for x, which is contained in s.
* $subs_{x,\psi}$ is a function that given a state $s \in Q^{(\psi)}$ and $e \in (\mathbb{N}_+ \cdot \{.\})^*$, it substitutes the set of guessed nodes for x, which is contained in s, with e.

• $\varphi = \exists X : \{\widetilde{\eta}\}.\ \psi$

$$accept_{\varphi}^{dec,int_{\mathcal{F}}}(q,t) = \bigvee_{s\in q} accept_{\psi}^{dec,int_{\mathcal{F}}}(s,t)$$

Chapter 8

Implementation

In this chapter we discuss how we implement the inductive construction of tree automata using our proposal from the previous chapter. Since the implementation will be integrated into the existing generator system, in order to be compatible to this system, the implementation is also realized using the functional programming language Haskell.

The implementation aims to produce a Haskell code which contains the transition functions and the accepting functions from the given set of MSO* formulas. To produce the transitions functions for a formula, additionally the sorted ranked alphabet and the free node variable (together with its type) should also be given. In the implementation, the transition functions of a formula are produced by first applying the inductive construction of the representation transitions until the final representation transitions are obtained. Then, from this final representation transitions, their representation input symbols are unfolded to get the transition functions in Haskell code.

In contrast to the construction of the Haskell code for the transition functions, the Haskell code of an accepting function can be directly produced by unfolding the inductive structure of the formula. This means, besides producing the Haskell code of the formula, the Haskell code of all subformulas should also be produced.

In this work we have not considered the refinement of the power set construction, which is needed to produce the representation deterministic transitions of a quantified formula in which an attribute formula is involved and one of the node variable in this attribute formula is the quantified variable. This has also not been considered in the accepting function.

8.1 The Modules

The implementation is structured into eight modules, where the dependencies between modules are given in figure 8.1. The arc from module A to module B means that module A depends on module B. We shall discuss each modules in the following subsections.

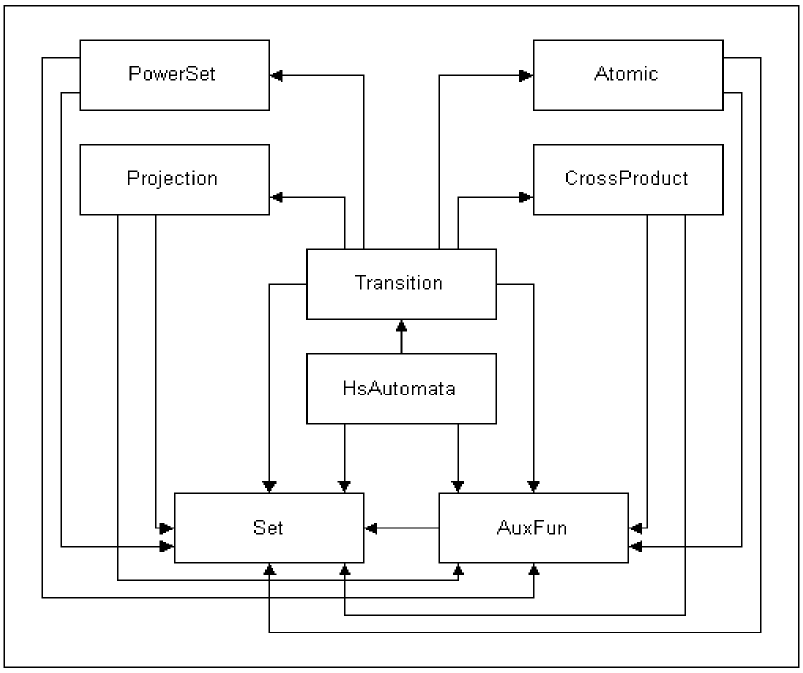

Figure 8.1: Dependencies between modules

8.1.1 Module *Set*

This module realizes the implementation of set, which is used in all modules. Here we represent a set as a binary search tree with no nodes having the same value, where we can perform the search and the insert operation of an element in $O(log\ n)$ (with n is the number of the nodes). It is defined by the following algebraic data type:

```
data Stree a = Null | Fork (Stree a) a (Stree a)
```

By representing a set using a binary search tree, we always keep the elements of a set to be ordered. The equality of two sets are defined by the equality of their sorted lists, where each sorted list is derived from the binary search tree (corresponds to each set) by in-order traversal. Similarly, the order of two sets are defined by the order of their sorted lists. In this module we also define some operations on sets, for instance element insertion, element deletion, set union, set difference, set intersection, sets cross product, power set operation, and map function on a set.

8.1.2 Module *AuxFun*

This module consists of the definition of all algebraic data types and the auxiliary functions which are used to realize the inductive construction of tree automata. Some important algebraic data types are the following:

- `Term`, which defines all terms from which an attribute formula can be formed:

```
data Term =
    FUNC FunName [Term]
  | ATTINST AttName Var1Name [Term]
```

- `Formula`, which defines the MSO* formulas:

```
data Formula =
    CONST Bool
  | LABEL ConName Var1Name
  | ELEM Var1Name Var2Name
  | EQUAL Var1Name Var1Name
  | TYPE TypName Var1Name
  | EDGE Int Var1Name Var1Name
  | OVER Var1Name Var1Name
  | ATTFOR Term
  | NEG Formula
  | AND Formula Formula
  | EXIST1 Var1Name (Set TypName) Formula
  | EXIST2 Var2Name (Set TypName) Formula
```

- `FormulaInfo`, which defines the information needed for constructing the transition functions and the accepting function of a formula:

```
data FormulaInfo = FI FormulaName PairVarTyp Formula
```

  where `FormulaName` is the name of a formula `Formula` and `PairVarTyp` is a pair of free node variable and its type.

- `Component` and `Ndum`, which together define the components of a state of an attribute formula (symbolic representation of a node):

```
data Component =
    DUM
  | NDUM Ndum Var1Name

data Ndum =
    EPS
  | PREF Int
  | PURE
```

  where `DUM` represents a dummy value and `NDUM` represents a Dewey notation symbolically for a node variable `Var1Name`, by one of the three constructors of the data type `Ndum`:

  – `EPS` for particular representation of the Dewey notation ε,

  – `PREF` for particular representation of Dewey notation with prefix (given by its integer argument),

  – `PURE` for general representation of Dewey notation.

- `RepVar1` and `RepVar2`, which define the representation node variable and node set variables, respectively:

```
data RepVar1 =
    NORM1 Var1Name
  | BAR1 Var1Name

data RepVar2 =
    NORM2 Var2Name
  | BAR2 Var2Name
```

where `NORM1` is a constructor denoting the node variable `Var1Name` without overline and `BAR1` is a constructor denoting the node variable `Var1Name` with overline. The constructors in `RepVar2` also have similar meaning.

- `State`, which defines the state of a tree automaton:

```
data State =
    SINT Int
  | SDEW [Component]
  | SAND State State
  | SEXT1 State Flag
  | SNON (Set State)
```

where `SINT` is a constructor denoting the state of all atomic formulas, except the attribute formula (whose state is denoted by the constructor `SDEW`), `SAND` is a constructor denoting the state of a formula with conjunction, `SEXT1` is a constructor denoting the state for a formula with quantifier over node variable (regarding its nondeterministic automaton), and `SNON` is a constructor denoting the state for a formula with quantifier (regarding its deterministic automaton).

- `Trans`, which defines the representation transitions:

```
data Trans =
    DT Rank Symbol InSt DetOutSt
  | NDT Rank Symbol InSt NDetOutSt
```

with:

```
type Rank = Int
type Symbol = (Set ConName,Set RepVar1,Set RepVar2)
type InSt = [State]
type DetOutSt = State
type NDetOutSt = State
```

The sorted ranked alphabet is defined in this module as the following type synonym:

```
type RankAlph = Set (Rank,ConName,ConType)
```

In this module, we also define auxiliary functions, which are independent of a particular construction. This means, they are defined so that they are reusable in many constructions. For example, the diamond operator (cf. subsection 7.3.2) and the compression procedure (cf. subsection 7.3.3) are implemented in this module.

8.1.3 Module *Atomic*

In this module we implement the construction of representation transitions for every atomic formula. The implementation is realized by following the formal construction which was given in subsection 7.3.1. For every atomic formula, several functions are defined. These functions corresponds to the cases described in the formal construction, where we consider, with respect to our assumption, for every possible configuration of input states, all possible representation input symbols which are complete and non-overlapping.

8.1.4 Module *CrossProduct*

This modules realizes the cross product construction for producing the representation transitions of formulas using conjunction, i.e $(\psi_1 \wedge \psi_2)$. The main access to the construction is given by the following function:

```
cross_reach :: Set Rank -> Set Trans -> Set Trans ->
               Set Trans -> Set State -> Set Trans -> Set State ->
               (Set Trans,Set State)
```

The function `cross_reach` has the following arguments:

- The first argument is the set of rank of input symbols from the ranked alphabet for which the representation transitions are constructed.
- The second argument is the set of representation transitions of ψ_1.
- The third argument is the set of representation transitions of ψ_2.
- The fourth argument is the set of representation transitions of $(\psi_1 \wedge \psi_2)$ constructed so far, cf. the role of T_{cr} in subsection 7.3.4.
- The fifth argument is the set of states reachable so far, cf. the role of S_{cr} in subsection 7.3.4.
- The sixth argument is the set of representation transitions of $(\psi_1 \wedge \psi_2)$, which are newly constructed, cf. the role of ΔT_{cr} in subsection 7.3.4.
- The seventh argument is the set of new reachable states, cf. the role of ΔS_{cr} in subsection 7.3.4.

The output of the function is a pair, where the first component of the pair is the set of all representation transitions of $(\psi_1 \wedge \psi_2)$ and the second component of the pair is the set of all reachable states.

8.1.5 Module *Projection*

In this module we implement the projection construction for producing the representation nondeterministic transitions of formulas using quantifier. The implementation in this module is divided into two parts:

1. The implementation of the projection construction for formulas with quantifier over a node variable, i.e. $(\exists x : \widetilde{\eta}.\ \psi)$.
   The main access to this construction is given by the following function:

```
proj1_reach :: PairVarTyp -> Set Trans -> RankAlph ->
               Set Trans -> Set State -> Set Trans -> Set State ->
               (Set Trans,Set State)
```

The function `proj1_reach` has the following arguments:

- The first argument is the pair of the quantified variable and its type. The type of the quantified variable is particularly needed considering case 2 of the construction described in page 95 and page 96.
- The second argument is the set of representation transitions of ψ.
- The third argument is the ranked alphabet. It is also needed considering case 2 of the construction described in page 95 and page 96.
- The fourth argument is the set of representation transitions for the formula $(\exists x : \widetilde{\eta}.\ \psi)$ constructed so far (the same role as T_{pr} in subsection 7.3.5).
- The fifth argument is the set of states reachable so far (the same role as S_{pr} in subsection 7.3.5).
- The sixth argument is the set of representation transitions for the formula $(\exists x : \widetilde{\eta}.\ \psi)$, which are newly constructed (the same role as ΔT_{pr} in subsection 7.3.5).
- The seventh argument is the set of new reachable states (the same role as ΔS_{pr} in subsection 7.3.5).

The output of the function is a pair, where the first component of the pair is the set of all representation transitions of $(\exists x : \widetilde{\eta}.\ \psi)$ and the second component of the pair is the set of all reachable states.

2. The implementation of the projection construction for formulas with quantifier over a node set variable, i.e. $(\exists X : \widetilde{\eta}.\ \psi)$.
   The main access to this construction is given by the following function:

   ```
   proj2_reach :: Var2Name -> Set Trans ->
                  Set Trans -> Set State -> Set Trans -> Set State ->
                  (Set Trans,Set State)
   ```

   In general, it has the same argument as the function `proj1_reach`, except that in this construction the type of the quantified variable (where for function `proj1_reach`, it is included in its first argument) and the ranked alphabet are not needed. The output is also a pair, where the first component of the pair is the set of all representation transitions of $(\exists X : \widetilde{\eta}.\ \psi)$ and the second component of the pair is the set of all reachable states.

8.1.6 Module *PowerSet*

This modules realizes the power set construction for determining the representation nondeterministic transitions resulting from the projection construction. The main access to the construction is given by the following function:

```
power_reach :: Set Rank -> Set Trans ->
               Set Trans -> Set State -> Set Trans -> Set State ->
               (SetTrans,Set State)
```

The function `power_reach` has the following arguments:

- The first argument is the set of rank of input symbols from the ranked alphabet for which the representation deterministic transitions are constructed.
- The second argument is the set of representation nondeterministic transitions which have to be determinized.
- The third argument is the set of representation deterministic transitions constructed so far (the same role as T_{ps} in subsection 7.3.6).
- The fourth argument is the set of states reachable so far (the same role as S_{ps} in subsection 7.3.6).
- The fifth argument is the set of representation deterministic transitions, which are newly constructed (the same role as ΔT_{ps} in subsection 7.3.6).
- The sixth argument is the set of new reachable states (the same role as ΔS_{ps} in subsection 7.3.6).

The output of the function is a pair, where the first component of the pair is the set of all representation deterministic transitions and the second component of the pair is the set of all reachable states.

8.1.7 Module *Transition*

In this module we define a function which realizes the inductive construction of representation transitions for every formula. This is performed by calling the functions which are defined in the following module:

- Module *Atomic*, for constructing the representation transitions of every atomic formula.
- Module *CrossProduct*, for constructing the representation transitions of formulas with conjunction.
- Module *Projection* and *PowerSet*, for constructing the representation transitions of formulas with quantifier (over node variable and node set variable).

The function for the compression of representation transitions (which is defined in the module *AuxFun*) is called after the cross product construction, the projection construction, and the power set construction are performed.

The function which produces the representation transitions for every formula is defined as the following:

```
dtrans :: Formula -> Set PairVarTyp -> RankAlph -> Set Trans
```

It has the following arguments:

- The first argument is the formula for which the representation transitions are computed.
- The second argument is a set of free variables and their types. Note that a free variable and its type are encapsulated in a pair with the type `PairVarTyp`. In function `dtrans`, we need a set of such pairs as an input, since in general we can have more than one free variable, for instance in the formula $edge_i(x, y)$ and $(x\ over\ y)$.
- The third argument is the ranked alphabet, which is needed to build the representation input symbol of a representation transition.

The output of this function is a set of representation transitions.

8.1.8 Module *HsAutomata*

This module realizes the Haskell code generation for the transition functions and accepting functions of a given list of formulas. Additionally, the algebraic data types needed by the transition functions and the accepting functions are also given in the output Haskell code.

The code of the transition functions are produced by first having a function call to function `dtrans` (which is defined in the module `Transition`) to perform the inductive construction of representation transitions from which the code of the transition functions are produced. Since in the representation transition we use a representation input symbol, then to have the input symbol of a transition function, we need to extract it from the representation input symbol.

The code of the accepting function of a formula is constructed by first extracting all its subformulas and give every subformulas a unique name (which is realized by giving an index according to the position of the subformula in the formula). Then, the Haskell code is produced, by following the definition of the accepting function in section 7.4, for the formula and for every of its subformulas.

As the main function of the module, we define the following:

```
prn_Automata :: RankAlph -> [FormulaInfo] -> String
```

It has the following arguments:

- The first argument is the ranked alphabet, which is particularly needed for producing the transition functions.
- The second argument is a list of formula information (see subsection 8.1.2 for how the algebraic data type `FormulaInfo` is defined).

The Haskell code is then produced by writing the output of this function into a file (which is performed by the built-in function `writeFile`).

8.2 Example of The Output

As an example of how the output of the implementation looks like, consider again the values of N, Σ, $\mathcal{V}_1$, $\mathcal{V}_2$, K, $\mathcal{A}$, and $\mathcal{F}$ defined in page 37. Consider also the following formula (which tests whether an identifier is in the declaration part):

$$\psi = \exists x : \{Decl\}.((x\ over\ y) \wedge (label_{DECL}(x))) \in MSO^*(\Sigma, \mathcal{V}_1 \setminus \{x^{\{Decl\}}\}, \mathcal{V}_2).$$

To produce the Haskell code, function `prn_Automata` is called with the following arguments:

- The first argument:

  ```
  ls [(0,"IDENT",["Ident"]),(0,"BOOLTYP",["TypExp"]),
      (0,"INTTYP",["TypExp"]),(1,"DSINGLE",["Decl","DList"]),
      (2,"DPAIR",["Decl","DList","DList"]),
      (2,"DECL",["Ident","TypExp","Decl"])]
  ```

  Note: function `ls :: (Ord a) => [a] -> Stree a` is a function defined in module *Set* which converts a list (not necessarily ordered) into a set.

- The second argument:

  ```
  [FI "psi" ("y",ls ["Ident"])
      (EXIST1 "x" (ls ["Decl"])
              (AND (OVER "x" "y") (LABEL "DECL" "x")))]
  ```

As the output, we obtain the transition functions and the accepting function for the formula ψ, which are written into a file (by built-in function `writeFile`):

```
-----------------------
-- ALGEBRAIC DATA TYPE
-----------------------
data State = SI Int
           | SD [Component]
           | SA State State
           | SE State Flag
           | SN [State]
           deriving (Eq,Show,Read)

data Component = D | ND Dewey
                 deriving (Eq,Show,Read)

data Dewey = EPS | DOT Int Dewey
             deriving (Eq,Show,Read)

data Flag = B | T
            deriving (Eq,Show,Read)


-----------------------------
-- TRANSITION FUNCTIONS: psi
-----------------------------
dlt_psi_BOOLTYP :: State
dlt_psi_BOOLTYP
  = (SN [(SE (SA (SI 0) (SI 0)) B)])
```

```
dlt_psi_IDENT_y :: State
dlt_psi_IDENT_y
  = (SN [(SE (SA (SI 1) (SI 0)) B)])

dlt_psi_IDENT :: State
dlt_psi_IDENT
  = (SN [(SE (SA (SI 0) (SI 0)) B)])

dlt_psi_INTTYP :: State
dlt_psi_INTTYP
  = (SN [(SE (SA (SI 0) (SI 0)) B)])

dlt_psi_DSINGLE :: State -> State
dlt_psi_DSINGLE (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 0) (SI 0)) B)])

dlt_psi_DSINGLE (SN [(SE (SA (SI 0) (SI 0)) B),
                     (SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])

dlt_psi_DSINGLE (SN [(SE (SA (SI 1) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B)])

dlt_psi_DSINGLE (SN [(SE (SA (SI 1) (SI 0)) B),
                     (SE (SA (SI 1) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])

dlt_psi_DSINGLE (SN [(SE (SA (SI 1) (SI 0)) B),
                     (SE (SA (SI 1) (SI 1)) T),
                     (SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DSINGLE (SN [(SE (SA (SI 1) (SI 0)) B),
                     (SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL :: State -> State -> State
dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B)])
             (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B)])
             (SN [(SE (SA (SI 1) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B)])
             (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])
```

```
dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
             (SN [(SE (SA (SI 1) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
             (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B)])
             (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B)])
             (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B)])
             (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B)])
             (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                  (SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B)])
             (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
```

```
dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
             (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                  (SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
             (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                  (SE (SA (SI 2) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                  (SE (SA (SI 2) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DECL (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
             (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR :: State -> State -> State
dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B)])
              (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B)])
              (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                   (SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B)])
              (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
```

```
dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
              (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                   (SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
              (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                   (SE (SA (SI 2) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
                   (SE (SA (SI 2) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 2) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T),
         (SE (SA (SI 2) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B)])
              (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 0) (SI 0)) B)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B)])
              (SN [(SE (SA (SI 1) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B)])
              (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
              (SN [(SE (SA (SI 1) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
              (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
```

```
dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B)])
              (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B)])

dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B)])
              (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])

dlt_psi_DPAIR (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])
              (SN [(SE (SA (SI 0) (SI 0)) B),(SE (SA (SI 0) (SI 1)) T)])
  = (SN [(SE (SA (SI 1) (SI 0)) B),(SE (SA (SI 1) (SI 1)) T)])

----------------------------
-- ACCEPTING FUNCTIONS: psi
----------------------------
accept_psi :: State -> Tree -> Bool
accept_psi (SN ls) t
  = foldr (\(SE st fl) y -> y || ((fl == T) && (accept_psi_1 st t)))
          False ls

accept_psi_1 :: State -> Tree -> Bool
accept_psi_1 (SA s1 s2) t
  = (accept_psi_1_1 s1 t) && (accept_psi_1_2 s2 t)

accept_psi_1_1 :: State -> Tree -> Bool
accept_psi_1_1 (SI s) _
  = (s == 2)

accept_psi_1_2 :: State -> Tree -> Bool
accept_psi_1_2 (SI s) _
  = (s == 1)
```

Chapter 9

Conclusion and Future Works

In this thesis we have discussed a variant of monadic second order (MSO) logic over trees, namely MSO* logic. The aim of having this logic is to allow the specifier of a language-based editor to specify the context-sensitive syntax of the language (for which the editor is generated) in a more comfortable way, since the specification can be made logically and globally. To achieve this aim, additional atomic formulas which are appropriate for specifying context-sensitive phenomena are introduced. In particular, we defined a so-called attribute formula which relates the attribute grammar formalism and the MSO logic formalism. By having this formula, one is then able to test a more subtle property between nodes which are expressed by their attributes. To establish the relationship between the attribute grammar formalism and the MSO logic formalism, an interface between them are defined in this thesis formally.

In order to integrate the MSO* logic formalism into the existing generator system [2], we need to define an operational semantics of this logic, where we use bottom up tree automata. We have generalized for MSO* logic the known transformation of an MSO formula into a bottom up tree automaton (which recognizes the tree language of this formula) [5, 13]. It turns out, that the transformation of an attribute formula yields a bottom up tree automaton with infinite number of states. This infinity issue comes from the fact, that in the states of the automaton for this formula, we collect the value of node variables which occur in the formula. However, the size of the tree from which the nodes are collected is unknown in the transformation step. Since the automaton of a formula is constructed inductively, we need to define in general a tree automaton with infinite number of states. In this thesis, we proposed a definition of a bottom up tree automaton with infinite number of states. For such automata, a family of sets of final states is needed, instead of just a set of final states as in the bottom up finite state tree automata. Every set of final state in this family is defined for the decoration of a tree on which the automaton runs.

Since the transformation of an MSO* formula into a tree automaton is realized by an inductive construction of tree automata, it is important to have an automaton with less number of states and less number of transitions. To achieve this, we consider several ways. First, in the transformation we make an assumption that the trees on which the automaton runs are well-marked trees. Second, we consider the sort information of symbols which label the nodes of a tree. Third, we develop a representation of tree automata which allows us to combine some transitions which have the same behavior. Additionally, this representation is also able to deal with the infinite number of states of the automata. Fourth, we consider only the reachable states during the inductive construction of tree automata.

In our representation of tree automata, the infinite number of states is handled by representing the states symbolically. In particular, this symbolic representation is used to represent the value of node variable in an attribute formula. Apparently, this symbolic representation is used only to represent a node in a tree, but it turns out, that this symbolic representation should also be able to represent a set of node. This is especially true, in case we consider a formula with quantifier, where an attribute formula occurs in the subformula and one of the node variables in this attribute formula is the quantified variable. In this case, the symbolic representation represents a set of guessed nodes for the quantified variable.

By introducing a representation of input symbols, we are able to minimize the number of representation of transitions. This representation of input symbols allows the transitions which have the same behavior to be represented as one transition. In defining the construction of the representation of transitions for atomic formulas, we give complete and non-overlapping cases with respect to our assumption. From these cases, it often happens that some transitions can be represented as one transition (by means of our representation of input symbols) because of their same behavior.

We also define the cross product construction, the projection construction, and the power set construction, which works on our representation. Generally, all of these constructions are defined based on the original construction. Particularly, in all of these constructions, we only consider the states which are reachable. With the existence of the attribute formula in our logic, the reachability construction is no more straightforward. A special handling for computing the reachable states and the transitions is needed, especially regarding the symbolic representation.

The examples of our proposed inductive construction of tree automata in this report give a picture of how the reduction of the number of states and the number of transitions can be achieved compared to the naive construction (where we consider every input symbol and every state as the input states of transitions).

In this work, generally we have implemented the approach we proposed for the inductive construction of tree automata. Nevertheless, the refinement of the power set construction, whose idea was given in the last part of the subsection 7.3.6 has not been implemented yet. This refinement mainly concerns with the determinization of the representation of nondeterministic transitions for a formula with quantifier where an attribute formula occurs in the subformula and one of the node variables in this attribute formula is the quantified variable. The accepting function for this case, whose idea was proposed in section 7.4, also need to be implemented. The completion of the implementation can be considered as one of the future works.

Another investigation would be to think about the possibility for renaming the states. In our work, we keep the structure of the states for some reasons.

By keeping the structure of the states, we can define the accepting function of a formula without any need of having the representation of the set of final states. Instead, it can be defined inductively by also giving the definition of the accepting functions for its subformulas. Since we keep the structure of the state (which also reflects the structure of the formula), the decision whether a state is a final state can be drawn by applying the definition of the accepting functions to the state inductively. In fact, if we represent the state symbolically, it is not possible to completely rename the state, since the symbolic representation acts as a parameter or a placeholder whose concrete value is only known after we let the automaton run on a tree. Hence, the occurrence of symbolic representation in a state should be kept. If the state renaming is considered, then one possibility would be to rename the state only up to non-symbolic representation.

In our work, we also do not consider the minimization algorithm to minimize the automata. With the existence of the attribute formula in our logic, the minimization algorithm apparently can not be performed. Generally, to perform the minimization algorithm, the set of final states should be clearly represented. On the other hand, to deal with the infinite number of states, we do not use the concrete value of states, but instead we represent the states symbolically. Particularly, this also holds for the final states. If an attribute formula is involved in a formula, we would not have the set of final states with concrete values in the inductive construction of the set of final states. One approach would be to minimize the automaton in case the attribute formula is not yet involved. For example, regarding a built formula, the minimization can be performed on a subformula, which contains no attribute formula, before the subsequent inductive construction (which involves an attribute formula) is performed.

Bibliography

[1] M. Biehl, N. Klarlund, and T. Rauhe. Algorithms for guided tree automata. In *First International Workshop on Implementing Automata, WIA'96, Lecture Notes in Computer Science, 1260.* Springer Verlag, 1996.

[2] E. Bormann. *Entwurf und Implementierung eines Systems zur Erzeugung syntaxgesteurter Editoren.* Master's Thesis, Department of Computer Science, Dresden University of Technology, 2000.

[3] E. Bormann. *Automatic Generation of Language-based Editors using the DSL.* Manuscript, Department of Computer Science, Dresden University of Technology, 2003.

[4] Z. Fülöp and H. Vogler. *Syntax-directed semantics — Formal models based on tree transducers.* Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1998.

[5] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[6] A. Kühnemann and H. Vogler. Synthesized and inherited functions — a new computational model for syntax-directed semantics. *Acta Inform.*, 31:431-477, 1994.

[7] N. Klarlund. Mona & Fido: The Logic-Automaton Connection in Practice. In *Computer Science Logic, 11^{th} International Workshop, CSL'97*, volume 1414 of *LNCS*, pages 311-326, 1997.

[8] N. Klarlund, A. Møller, and M.I. Schwartzbach. MONA Implementation Secrets, to appear in Proc. *International Journal of Foundation of Computer Science*, 2000. CIAA 2000.

[9] F. Morawietz and T. Cornell. On the recognizability of relations over a tree definable in a monadic second order tree description language. Technical Report SFB 340, Seminar für Sprachwissenschaft Eberhard-Karls-Universität Tübingen, 1997.

[10] A.R. Meyer. Weak monadic second-order theory of successor is not elementary recursive. In R. Parikh, editor, *Logic Colloquium, (Proc. Symposium on Logic, Boston, 1972)*, volume 453 of *LNCS*, pages 132-154, 1975.

[11] M. Steinmann. Übersetzung von logischen Ausdrücken in Baumautomaten: Entwicklung eines Verfahrens und seine Implemetierung. Master's Thesis,

Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Uni. zu Kiel, 1993.

[12] L. Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, Dept. of Electrical Eng., M.I.T., Cambridge, MA, 1974. Report TR-133.

[13] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with application to a decision problem of second-order logic. *Math. Systems Theory*, 2(1):57-81, 1968.

[14] J.D. Ullman. *Principles of Database and Knowledge-base systems*, Vol. 1, Computer Science Press, 1988.

Statement of Academical Honesty

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Date: March 9th, 2004
Signature: