Manual

Shuai Yuan

The program "translation" parse SQL phrases into Why3ML program. It mainly contains the following parts:

- header of Why3ML program, mainly commands of importing modules.
- parser for the SQL table definition, this part translate the "CREATE TABLE" phrase into the definition of the type of the corresponding table's tuple.
- parser for the SQL assertion, this part translate the "CREATE ASSERTION" phrase into "predicate" in the Why3ML program.
- parser for the SQL INSERT command, this part translate the SQL INSERT command into a method in the Why3ML program.
- parser for the SQL DELETE command, this part translate the SQL INSERT command into a method in the Why3ML program.
- parser for the SQL UPDATE command, this part translate the SQL INSERT command into a method in the Why3ML program.

1 Parser for the SQL Table Definition

The source language of SQL table definition is expressed in the following grammar:

<table definition>	::=	CREATE TABLE <*table name*>
		(<table element list>)
<table element list>	::=	<table element>
	|	<table element list>, <table element>
<table element>	::=	<*column name*> <data type>
<data type>	::=	INTEGER
		SMALLINT
		FLOAT
		NUMERIC
		BOOLEAN

2 Parser for the SQL Assertion

The grammar of SQL assertion is:

CREATE ASSERTION <*assertion name*>
CHECK <exists predicate>

$$
\begin{array}{rcl}
\langle\text{exists predicate}\rangle & ::= & [\ \text{NOT}\ ]\ \text{EXISTS}\ (\ \langle\text{query expression}\rangle\ ) \\
\langle\text{query expression}\rangle & ::= & \text{SELECT *} \\
 & & \text{FROM}\ \langle\text{table list}\rangle \\
 & & \text{WHERE}\ \langle\text{search condition}\rangle \\
\langle\text{table list}\rangle & ::= & \langle\textit{table name}\rangle\ \langle\textit{tuple name}\rangle \\
 & & \langle\text{table list}\rangle, \langle\textit{table name}\rangle\ \langle\textit{tuple name}\rangle \\
\langle\text{search condition}\rangle & ::= & \langle\text{boolean term}\rangle \\
 & | & \langle\text{search condition}\rangle\ \text{OR}\ \langle\text{boolean term}\rangle \\
\langle\text{boolean term}\rangle & ::= & \langle\text{boolean factor}\rangle \\
 & | & \langle\text{boolean term}\rangle\ \text{AND}\ \langle\text{boolean factor}\rangle \\
\langle\text{boolean factor}\rangle & ::= & \langle\text{predicate}\rangle \\
 & | & [\ \text{NOT}\ ]\ (\ \langle\text{search condition}\rangle\ ) \\
\langle\text{predicate}\rangle & ::= & \langle\text{exists predicate}\rangle \\
 & | & \langle\text{comparison predicate}\rangle \\
 & | & \langle\text{between predicate}\rangle \\
 & | & \langle\text{in predicate}\rangle \\
 & | & \langle\text{null predicate}\rangle \\
\langle\text{comparison predicate}\rangle & ::= & \langle\text{expression}_1\rangle\ \langle\text{comp op}\rangle\ \langle\text{expression}_2\rangle \\
\langle\text{comp op}\rangle & ::= & =\ |\ \text{<>}\ |\ <\ |\ \leq\ |\ >\ |\ \geq \\
\langle\text{expression}\rangle & ::= & \langle\text{term}\rangle \\
 & | & \langle\text{expression}\rangle\ \{+ \mid -\}\ \langle\text{term}\rangle \\
\langle\text{term}\rangle & ::= & \langle\text{factor}\rangle \\
 & | & \langle\text{term}\rangle\ \{* \mid /\}\ \langle\text{factor}\rangle \\
\langle\text{factor}\rangle & ::= & (\langle\text{expression}\rangle) \\
 & | & [+ \mid -]\langle\textit{const}\rangle \\
 & | & [+ \mid -]\langle\text{column}\rangle \\
\langle\text{column}\rangle & ::= & \langle\textit{tuple name}\rangle.\langle\textit{attribute name}\rangle \\
\langle\text{between predicate}\rangle & ::= & \langle\text{expression}\rangle\ [\ \text{NOT}\ ] \\
 & & \text{BETWEEN}\ \langle\textit{const}_1\rangle\ \text{AND}\ \langle\textit{const}_2\rangle \\
\langle\text{in predicate}\rangle & ::= & \langle\text{expression}\rangle[\ \text{NOT}\ ]\ \text{IN}\ (\ \langle\text{in value list}\rangle\ ) \\
\langle\text{in value list}\rangle & ::= & \langle\textit{const}\rangle \\
 & | & \langle\text{in value list}\rangle, \langle\textit{const}\rangle \\
\langle\text{null predicate}\rangle & ::= & \langle\text{column}\rangle\ \text{IS}\ [\ \text{NOT}\ ]\ \text{NULL}
\end{array}
$$

The general form of the target Why3ML code is:

$$
\begin{array}{c}
\text{predicate}\ \langle\text{assertion name}\rangle\ \langle\text{parameters}\rangle = \\
\langle\text{logical formula}\rangle
\end{array}
$$

We define the function $\mathcal{T}$ as the translational function mapping a SQL assertion phrase into a logical formula.

$$
\begin{array}{rlcl}
\mathcal{T}[\text{CREATE ASSERTION}\ \langle\text{assertion name}\rangle & & & \\
\text{CHECK}\ \langle\text{exists predicate}\rangle] & & \rightsquigarrow & \mathcal{T}[\langle\text{exists predicate}\rangle] \\
\mathcal{T}[\text{EXISTS}\ (& \text{SELECT *} & \rightsquigarrow & \text{exists}\ x_1, \ldots, x_n. \\
 & \text{FROM}\ R_1\ x_1, \cdots, R_n\ x_n & & x_1 \in R_1 \land \ldots \land x_n \in R_n \\
 & \text{WHERE}\ \langle\text{search condition}\rangle\ )] & & \land\mathcal{T}[\langle\text{search condition}\rangle] \\
\mathcal{T}[\text{NOT EXISTS}\ (& \text{SELECT *} & \rightsquigarrow & \text{not (exists}\ x_1, \ldots, x_n. \\
 & \text{FROM}\ R_1\ x_1, \cdots, R_n\ x_n & & x_1 \in R_1 \land \ldots \land x_n \in R_n \\
 & \text{WHERE}\ \langle\text{search condition}\rangle\ )] & & \land\mathcal{T}[\langle\text{search condition}\rangle])
\end{array}
$$

$$\begin{array}{rcl}
\mathcal{T}[\text{<search condition> OR <boolean term>}] & \rightsquigarrow & \mathcal{T}[\text{<search condition>}] \lor \mathcal{T}[\text{<boolean term>}] \\
\mathcal{T}[\text{<boolean term> AND <boolean factor>}] & \rightsquigarrow & \mathcal{T}[\text{<boolean term>}] \land \mathcal{T}[\text{<boolean factor>}] \\
\mathcal{T}[\text{NOT (<predicate>)}] & \rightsquigarrow & \neg\, (\mathcal{T}[\text{<predicate>}])
\end{array}$$

$$\begin{array}{rcl}
\mathcal{T}[\text{<expression}_1\text{ > <comp op> <expression}_2\text{ >}] & \rightsquigarrow & \mathcal{T}[\text{<expression}_1\text{ >}] \\
 & & \mathcal{T}[\text{<comp op>}]\ \mathcal{T}[\text{<expression}_2\text{ >}] \\
\text{<comp op>} & ::= & = \mid <> \mid < \mid \leq \mid > \mid \geq \\
\mathcal{T}[\text{<comp op>}] & \rightsquigarrow & \text{<comp op>} \\
\mathcal{T}[\text{<expression}_1\text{ > <numerical op> <expression}_2\text{ >}] & \rightsquigarrow & \mathcal{T}[\text{<expression}_1\text{ >}] \\
 & & \mathcal{T}[\text{<numerical op>}]\ \mathcal{T}[\text{<expression}_2\text{ >}] \\
\text{<numerical op>} & ::= & + \mid - \mid \times \mid / \\
\mathcal{T}[\text{<numerical op>}] & \rightsquigarrow & \text{<numerical op>} \\
\mathcal{T}[\text{<const>}] & \rightsquigarrow & \text{<const>} \\
\mathcal{T}[x.a] & \rightsquigarrow & x.a
\end{array}$$

$$\begin{array}{lcl}
\mathcal{T}[\text{<expression> BETWEEN} & \rightsquigarrow & (\mathcal{T}[\text{<expression>}]\ \geq\ \text{<const}_1\text{ >}) \\
\quad \text{<const}_1\text{ > AND <const}_2\text{ >}] & & \land\, (\mathcal{T}[\text{<expression>}]\ \leq\ \text{<const}_2\text{ >}) \\
\mathcal{T}[\text{<expression> NOT BETWEEN} & \rightsquigarrow & (\mathcal{T}[\text{<expression>}]\ <\ \text{<const}_1\text{ >}) \\
\text{<const}_1\text{ > AND <const}_2\text{ >}] & & \land\, (\mathcal{T}[\text{<expression>}]\ >\ \text{<const}_2\text{ >})
\end{array}$$

$$\begin{array}{lcl}
\mathcal{T}[\text{<expression> IN} & \rightsquigarrow & \text{mem } \mathcal{T}[\text{<expression>}] \\
\ (\ \text{<const}_1\text{ >}, \ldots, \text{<const}_m\text{ >}\ )\ ] & & (\text{Cons <const}_1\text{ > (Cons} \ldots (\text{Cons <const}_m\text{ > Nil}) \ldots)) \\
\mathcal{T}[\text{<expression> NOT IN} & \rightsquigarrow & \text{not (mem } \mathcal{T}[\text{<expression>}] \\
\ (\ \text{<const}_1\text{ >}, \ldots, \text{<const}_m\text{ >}\ )\ ] & & (\text{Cons <const}_1\text{ > (Cons} \ldots (\text{Cons <const}_m\text{ > Nil}) \ldots)))
\end{array}$$

$$\begin{array}{rcl}
\mathcal{T}[x.a \text{ IS NULL}] & \rightsquigarrow & x.a = \text{NULL} \\
\mathcal{T}[x.a \text{ IS NOT NULL}] & \rightsquigarrow & x.a \neq \text{NULL}
\end{array}$$

Let *exp* be a source language phrase, then:

$$\mathcal{T}[(exp)] \rightsquigarrow (\mathcal{T}[exp])$$

3 Parser for the SQL INSERT statement

The grammar of SQL insert statement is:

$$\begin{array}{rcl}
\text{<insert statement>} & ::= & \text{INSERT INTO <}\textit{table name}\text{> VALUES (<column value list>)} \\
 & \mid & \text{INSERT INTO <}\textit{table name}\text{> (<column name list>)} \\
 & & \text{VALUES (<column value list>)} \\
\text{<column name list>} & ::= & \text{<}\textit{column name}\text{>} \\
 & \mid & \text{<column name list>, <}\textit{column name}\text{>} \\
\text{<column value list>} & ::= & \text{<column value>} \\
 & \mid & \text{<column value list>, <}\textit{column value}\text{>}
\end{array}$$

The general form of the target Why3ML code is:

$$\begin{array}{rcl}
\text{<insert function>} & ::= & \text{let <}\textit{fun name}\text{> <}\textit{fun parameters}\text{> =} \\
 & & \{\ \text{<precondition>}\ \} \\
 & & \text{<}\textit{target table}\text{> ++ <new tuple>} \\
 & & \{\ \text{<postcondition>}\ \} \\
\text{<precondition>} & ::= & \text{<}\textit{assertion name}\text{> <}\textit{assertion arguments}\text{>} \\
 & \mid & \text{<precondition>} \land \text{<}\textit{assertion name}\text{> <}\textit{assertion arguments}\text{>} \\
\text{<new tuple>} & ::= & (\ \text{Cons } \{\mid \text{<new column list>} \mid\} \text{ Nil}\ ) \\
\text{<new column list>} & ::= & \text{<}\textit{column name}\text{> = <}\textit{column value}\text{>} \\
 & \mid & \text{<new column list>; <}\textit{column name}\text{> = <}\textit{column value}\text{>}
\end{array}$$

The grammar of <postcondition> is the same as that of <precondition> except that all occurrences of <target table> in the <assertion arguments> are replaced by “result”.

4 Parser for the SQL DELETE statement

The grammar of SQL delete statement is:

<delete statement>	::=	DELETE FROM <*target table name*>
		[USING <table reference list>]
		[WHERE <search condition>]
<table reference list>	::=	<*table name*>
	|	<table reference list>, <*table name*>
<search condition>	::=	<boolean term>
	|	<search condition> OR <boolean term>
<boolean term>	::=	<boolean factor>
	|	<boolean term> AND <boolean factor>
<boolean factor>	::=	<predicate>
	|	[NOT] (<search condition>)
<predicate>	::=	<comparison predicate>
	|	<between predicate>
	|	<in predicate>
	|	<null predicate>

The left parts are the same as those in the SQL assertion, so they are omitted in this manual.

If <search condition> is not specified in the delete statement, then the general form of the target Why3ML code is:

<delete function>	::=	let rec <*fun name*> <fun parameters> =
		{ true }
		match <*target table*> with
		| Nil → Nil
		| Cons {| <tuple exp> |} <*left table*> →
		(<*fun name*> <*fun arguments*>)
		end
		{ <postcondition> }
<tuple exp>	::=	<*column name*> = <column value string>
	|	<tuple exp>; <*column name*> = <column value string>
<column value string>	::=	<*table name*>_<*column name*>_value
<postcondition>	::=	<condition> → <consequence>
<condition>	::=	<*assertion name*> <*assertion arguments*>
	|	<condition> ∧ <*assertion name*> <*assertion arguments*>

The grammar of <fun arguments> is the same as that of <fun parameters> except that all occurrences of <target table> are replaced by <left table>. The grammar of <consequence> is the same as that of <condition> except that all occurrences of <target table> in the <assertion arguments> are replaced by “result”.

If <search condition> is specified and there is only one table in the delete statement, then the general form of the target Why3ML code is:

```
<delete function>  ::=  let rec <fun name> <fun parameters> =
                        { true }
                        match <target table> with
                        | Nil → Nil
                        | Cons {| <tuple exp> |} <left table> →
                        if <search condition> then ( <fun name> <fun arguments> )
                        else Cons {| <tuple exp> |} ( <fun name> <fun arguments> )
                        end
                        { <postcondition> }
```

If <search condition> is specified and more than one tables are involved in the delete statement, then we generate a predicate, a set of iteration functions and a delete function.

The predicate is used to represent the <search condition>, which will be used in the postcondition part of the iteration functions and the delete function. The general form of the predicate is:

```
<sc predicate>  ::=  predicate <predicate name> <predicate parameters> =
                     <search condition>
```

The iteration functions are used to obtain the required column values from tables other from the target table. The general form of the iteration function is:

```
<iter function>  ::=  let rec <fun name> <fun parameters> =
                      { <precondition> }
                      match <iter table> with
                      | Nil → False
                      | Cons {| <tuple exp> |} <left table> →
                      if <check condition> then True
                      else ( <fun name> <fun arguments> )
                      end
                      { <postcondition> }
<precondition>   ::=  <assertion name> <assertion arguments>
                 |    <precondition> ∧ <assertion name> <assertion arguments>
```

Let ITL be the list of tables that have been already iterated, then $\forall arg \in$ <assertion arguments>, $arg \in ITL$.

```
  <postcondition>   ::=  <precondition> ∧ ( result = True → <exists statement> )
<exists statement>  ::=  exists <tuple>: <tuple type>.
                         mem <tuple> <iter table> ∧ ( <sc predicate> <predicate arguments> )
```

The delete function is the function that will delete tuples from the target table. The general form of the delete function is:

<table>
<tr><td><delete function></td><td>::=</td><td>let rec <fun name> <fun parameters> =
{ <precondition> }
match <target table> with
| Nil → False
| Cons {| <tuple exp> |} <left table> →
if <check condition> then (<fun name> <fun arguments>)
else Cons {| <tuple exp> |} (<fun name> <fun arguments>)
end
{ <postcondition> }</td></tr>
</table>

5 Parser for the SQL UPDATE statement

The grammar of SQL update statement is:

<table>
<tr><td><update statement></td><td>::=</td><td>UPDATE <<i>target table name</i>>
SET <set clause list>
[FROM <table reference list>]
[WHERE <search condition>]</td></tr>
<tr><td><set clause list></td><td>::=</td><td><set clause>
<set clause list>, <set clause></td></tr>
<tr><td><set clause></td><td>::=</td><td><set column> = <<i>const</i>></td></tr>
<tr><td><set column></td><td>::=</td><td><<i>table name</i>>.<<i>attribute name</i>></td></tr>
</table>

The left parts are the same as those in the SQL delete statement grammar, so they are omitted in this manual.

If <search condition> is not specified in the update statement, then the general form of the target Why3ML code is:

<table>
<tr><td><update function></td><td>::=</td><td>let rec <fun name> <fun parameters> =
{ true }
match <target table> with
| Nil → Nil
| Cons {| <old tuple exp> |} <left table> →
Cons {| <new tuple exp> |} (<fun name> <fun arguments>)
end
{ <postcondition> }</td></tr>
<tr><td><tuple exp></td><td>::=</td><td><column name> = <column value string></td></tr>
<tr><td></td><td>|</td><td><tuple exp>; <column name> = <column value string></td></tr>
<tr><td><column value string></td><td>::=</td><td><table>_<column>_<value></td></tr>
<tr><td><postcondition></td><td>::=</td><td><condition> → <consequence></td></tr>
<tr><td><condition></td><td>::=</td><td><<i>assertion name</i>> <<i>assertion arguments</i>></td></tr>
<tr><td></td><td>|</td><td><condition> ∧ <<i>assertion name</i>> <<i>assertion arguments</i>></td></tr>
</table>

The grammar of <fun arguments> is the same as that of <fun parameters> except that all occurrences of <target table> are replaced by <left table>. The grammar of <consequence> is the same as that of <condition> except that all occurrences of <target table> in the <assertion arguments> are replaced by "result".

If <search condition> is specified and there is only one table in the update statement, then the general form of the target Why3ML code is:

```
<update function>  ::=  let rec <fun name> <fun parameters> =
                        { true }
                        match <target table> with
                        | Nil → Nil
                        | Cons {| <old tuple exp> |} <left table> →
                        if <search condition>
                        then Cons {| <new tuple exp> |} ( <fun name> <fun arguments> )
                        else Cons {| <old tuple exp> |} ( <fun name> <fun arguments> )
                        end
                        { <postcondition> }
```

If <search condition> is specified and more than one tables are involved in the update statement, then we generate a predicate, a set of iteration functions and a update function.

The predicate is used to represent the <search condition>, which will be used in the postcondition part of the iteration functions and the update function. The general form of the predicate is:

```
<sc predicate>  ::=  predicate <predicate name> <predicate parameters> =
                     <search condition>
```

The iteration functions are used to obtain the required column values from tables other from the target table. The general form of the iteration function is:

```
<iter function>  ::=  let rec <fun name> <fun parameters> =
                      { <precondition> }
                      match <iter table> with
                      | Nil → False
                      | Cons {| <tuple exp> |} <left table> →
                      if <check condition> then True
                      else ( <fun name> <fun arguments> )
                      end
                      { <postcondition> }
<precondition>   ::=  <assertion name> <assertion arguments>
                  |   <precondition> ∧ <assertion name> <assertion arguments>
```

Let ITL be the list of tables that have been already iterated, then $\forall arg \in$ <assertion arguments>$, arg \in ITL$.

```
   <postcondition>  ::=  <precondition> ∧ ( result = True → <exists statement> )
<exists statement>  ::=  exists <tuple>: <tuple type>.
                         mem <tuple> <iter table> ∧ ( <sc predicate> <predicate arguments> )
```

The update function is the function that will update tuples from the target table. The general form of the update function is:

<update function> ::= let rec <fun name> <fun parameters> =
{ <precondition> }
match <target table> with
| Nil → False
| Cons {| <old tuple exp> |} <left table> →
if <check condition>
then Cons {| <new tuple exp> |} (<fun name> <fun arguments>)
else Cons {| <new tuple exp> |} (<fun name> <fun arguments>)
end
{ <postcondition> }