Mapping Pre- and Post-conditions from OCL expressions to XPDDL and PDDL

Draft 2.0

Tiago Stegun Vaquero
Department of Mechanical & Industrial Engineering
University of Toronto
e-mail: tvaquero@mie.utoronto.ca

September 10, 2012

This document shows how the tool itSIMPLE maps pre- and post-condition written in OCL into XPDDL and PDDL in the action translation process. New OCL expressions capabilities will be available in further releases.

OCL Expression	XPDDL	PDDL
(1) *precond* or *poscond* expressions with logic operator "*and*" as following: `[expression1]` **`and`** `[expression2]` **`and ... and`** `[expression N]`	`<and>` `[expression1]` `[expression2]` … `[expressionN]` `</and>`	`(and` `[expression1]` `[expression2]` `...` `[expressionN]` `)`
(2) *precond* or *poscond* expressions with logic operator "*or*" as following: `[expression1]` **`or`** `[expression2]` **`or ... or`** `[expressionN]`	`<or>` `[expression1]` `[expression2]` … `[expressionN]` `</or>`	`(or` `[expression1]` `[expression2]` `...` `[expressionN]` `)`
(3) *precond* or *poscond* expressions as following: **`not`**`([expression])`	`<not>` `[expression1]` `</not>`	`(not([expression]))`
(4) Expression `[expression]` in *precond* in the following form: **`p0 = p1`** where: `p0` and `p1` are parameter of an action *act*. The case **`p0 <> p1`** is similar to `not(p0 = p1)`.	`<equals>` `<parameter id="p0">` `<parameter id="p0">` `</equals>`	`(= ?p0 ?p1)`
(5) A *precond* or *poscond* expression `[expression]` in the following form:	`<predicate id="attr">` `<parameter id="p0">` `</predicate>`	`(attr ?p0)`

`p0.attr = true` where: `p0` is a parameter of an action *act*; and `attr` is a non-parameterized boolean attribute of `p0` .		

(6) A *precond* or *poscond* expression `[expression]` in the following form:
`p0.attr = false`
where: `p0` is a parameter of an action *act*; and `attr` is a non-parameterized boolean attribute of `p0` .
	<not>
	  <predicate id="attr">
	    <parameter id="p0">
	     <parameter id="p1">
	  </predicate>
	</not>
		`(not (attr ?p0))`

(7) A *precond* or *poscond* expression `[expression]` in the following form:
`p0.attr(o1,...,on) = true`
where: `p0` is a parameter of an action *act*; and `attr` is a parameterized boolean attribute of `p0` . `o1,...,on` are parameters of `attr.` In the case of **`false`** value the translation is similar to `not([expression]).`
	<predicate id="attr">
	    <parameter id="p0">
	    <parameter id="o1">
	     ...
	    <parameter id="on">
	</predicate>
		`(attr ?p0 ?o1 ... ?on)`

(8) A *precond* or *poscond* expression `[expression]` in the following form:
`p0.attr = p1`
where: `p0` and `p1` are parameters of an action *act*; and `attr` is a non-parameterized non-primitive attribute In the case of "**<>**" logic operator, the translation is similar to `not([expression]).`
	<predicate id="attr">
	    <parameter id="p0">
	     <parameter id="p1">
	</predicate>
		`(attr ?p0 ?p1)`

(9) A *precond* or *poscond* expression `[expression]` in the following form:
`p0.attr(o1,...,on) = p1`
where: `p0` is a parameter of an action *act*; and `attr` is a parameterized non-primitive attribute of `p0.` `o1,...,on` are parameters of `attr.` In the case of **`false`** value the translatior is similar to `not([expression]).`
	<predicate id="attr">
	    <parameter id="p0">
	    <parameter id="o1">
	     ...
	    <parameter id="on">
	     <parameter id="p1">
	</predicate>
		`(attr ?p0 ?o1 ...`
		`     ?on ?p1)`

(10) A *precond* or *poscond* expression `[expression]` in the following form:
`p0.role = p1`
where: `p0` and `p1` are parameters of an action *act*; and `role` is a *rolename*, with multiplicity "1" or "0..1", of an associatio between the classes of `p0` and `p1`. In the case of "**<>**" operator, the translation is similar to `not([expression]).`
	<predicate id="role">
	    <parameter id="p0">
	    <parameter id="p1">
	</predicate>
		`(role ?p0 ?p1)`

(11) A *precond* expression `[expression]` in the following form: **`p0.role->exists(p: P|p = p1)`** where: `p0` and `p1` are parameters of an action *act*; and `role` is a *rolename*, with multiplicity greater than 2 or "*", of an association between the classes of `p0` e `p1`. **`P`** is the class of p1. In the case of "**<>**" operator, the translation is similar to `not([expression])`.	<predicate id="role"> <parameter id="p0"> <parameter id="p1"> </predicate>	`(role ?p0 ?p1)`
(12) A *precond* or *poscond* expression `[expression]` in the following form: **`attr = true`** where: `attr` is a *global* non-parameterized boolean attribute. An attribute is considered global when its class has stereotype Utility. In the case of **`false`** value the translation is similar to `not([expression])`.	<predicate id="attr" />	`(attr)`
(13) A *precond* or *poscond* expression `[expression]` in the following form: **`attr(o1,...,on) = true`** where: `attr` is a *global* parameterized boolean attribute. `o1,...,on` are parameters of the attribute `attr`. An attribute is considered global when its class has stereotype Utility. In the case of **`false`** value the translation is similar to `not([expression])`.	<predicate id="attr"> <parameter id="o1"> ... <parameter id="on"> </predicate>	`(attr ?o1 ... ?on)`
(14) A *precond* or *poscond* expression `[expression]` in the following form: **`attr = p1`** where: `attr` is a *global* non-parameterized non-primitive attribute and `p1` is a parameter of an action *act*. An attribute is considered global when its class has stereotype Utility. In the case of "**<>**" operator, the translation is similar to `not([expression])`.	<predicate id="attr"> <parameter id="p1"> </predicate>	`(attr ?p1)`
(15) A *precond* or *poscond* expression `[expression]` in the following form: **`attr(o1,...,on) = p1`** where: `attr` is a *global* parameterized non-primitive attribute and `p1` is a parameter of an action *act*. `o1,...,on`	<predicate id="attr"> <parameter id="o1"> ... <parameter id="on"> <parameter id="p1"> </predicate>	`(attr ?o1 ... ?on ?p1)`

are parameters of the attribute `attr`. An attribute is considered global when its class has stereotype Utility. In the case of "**<>**" operator, the translation is similar to `not([expression])`.		
(16) A *precond* or *poscond* expression `[expression]` in the following form: **`p0.role = p1`** where: `p0 and p1` are parameters of an action *act*; and `role` is a *rolename* of an association between the classes of `p0` and `p1`. In the case of "**<>**" operator, the translation is similar to `not([expression])`.	<predicate id="role"> <parameter id="p0"> <parameter id="p1"> </predicate>	`(role ?p0 ?p1)`

(17) Expressions that are equal to **`null`** or "->isEmpty()" in *precond* or *poscond* are ignored in the current version of itSIMPLE. For example
`p0.role = null`
or
`p0.role->isEmpty()`
	Ignored (but will be treated very soon)	`Ignored (but will be treated very soon)`

(18) Numeric expressions in *precond* are translated in the prefix form. For example
`p0.attr > p1.attr1 + num`
where: `p0` and `p1` are parameters of an action *act*; `attr` and `attr1` are numeric attributes and `num` is a Integer or a Float value.
Observation: in these expression it is possible to use operator such as: "=", ">=", "<=" e "<>" (i.e., not("=")). It is also possible to use "+", "-", "/" e "*".
	Example:
	<gt>
	<function id="attr">
	<parameter id="p0">
	</function>
	<add>
	<function id="attr1">
	<parameter id="p1">
	</function>
	<value number="num"/>
	</add>
	</gt>
		`Example:`
		`(> (attr ?p0)`
		`(+ (attr1 ?p1) num)`
		`)`

(19) Numeric expressions in *poscond* are translated in the prefix form where the operator "=" means value attribution. For example:
`p0.attr = p1.attr1 + num`
where: `p0` and `p1` are parameters of an action *act*; `attr` and `attr1` are numeric attributes and `num` is a Integer or a Float value.
	Example:
	<assign>
	<function id="attr">
	<parameter id="p0">
	</function>
	<add>
	<function id="attr1">
	<parameter id="p1">
	</function>
	<value number="num"/>
	</add>
	</assign >
		`Example:`
		`(= (attr ?p0)`
		`(+ (attr1 ?p1) num)`
		`)`

(20) Numeric expressions in *poscond* as following:
`p0.attr = p0.attr + num`
	Example:
	<increase>
	<function id="attr">
	<parameter id="p0">
		`Example:`
		`(increase (attr ?p0)`
		`num)`

where: p0 and p1 are parameters of an action *act*; attr is a numeric attribute and num is a Integer or a Float value.	</function> <value number=”num” /> </increase >	

(21) Numeric expressions in *poscond* as following:
p0.attr = p0.attr - num
where: p0 and p1 are parameters of an action *act*; attr is a numeric attribute and num is a Integer or a Float value.
	Example:
	<decrease>
	    <function id=”attr”>
	      <parameter id=”p0”>
	    </function>
	    <value number=”num” />
	</decrease >
		Example:
		(decrease (attr ?p0)
		    num)

(22) A *precond* expression [expression] in the following form:
p0.role->includes(p1)
where: p0 and p1 are parameters of an action *act*; and role is a *rolename*, with multiplicity greater than 2 or “*”, of an association between the classes of p0 e p1. In the case of **excludes** operator, the translation is similar to not([expression]).
	<predicate id=”role”>
	    <parameter id=”p0”/>
	    <parameter id=”p1”/>
	</predicate>
		(role ?p0 ?p1)

(23) A *postcond* expression [expression] in the following form:
p0.role = p0.role->including(p1)
where: p0 and p1 are parameters of an action *act*; and role is a *rolename*, with multiplicity greater than 2 or “*”, of an association between the classes of p0 e p1. In the case of **excluding** operator, the translation is similar to not([expression]).
	<predicate id=”role”>
	    <parameter id=”p0”/>
	    <parameter id=”p1”/>
	</predicate>
		(role ?p0 ?p1)

(24) A *precond* expression [expression] in the following form:
p0.role->forAll(p: P|
[expression2])
where: p0 is a parameter of an action *act*; and role is a *rolename*, with multiplicity greater than 2 or “*” of an association between the class of p0 e another class (**P**). **[expression2]** is another *precond* expression.
	<forall>
	   <parameter name=”p”
	type”p1”/>
	   <imply>
	      <predicate id=”role”>
	         <parameter id=”p0”/>
	         <parameter id=”p1”/>
	      </predicate>
	      <implies>
	         <<expression 2
	translation>>
	      </implies>
	   </imply>
	</forall>
		(forall (?p - p1)
		    (imply
		        (role ?p0 ?p)
		         <<expression2
		translation>>
		    )
		)

(25) A *postcond* expression [expression] in the following form:
p0.role->forAll(p: P |
	<forall>
	   <parameter name=”p”
	type”p1”/>
		(forall (?p - P)
		    (when
		        (role ?p0 ?p)

[expression2])
where: `p0` is a parameter of an action *act*; and `role` is a *rolename*, with multiplicity greater than 2 or “*”,of an association between the class of `p0` e another class (**P**). **[expression2]** is another *postcond* expression.
	<when>
	  <predicate id=”role”>
	    <parameter id=”p0”>
	    <parameter id=”p1”>
	  </predicate>
	  <do>
	    <<expression 2 translation>>
	  </do>
	</when>
	</forall>
		`<<expression2 translation>>`
		`)`
		`)`

(26) A *precond* expression `[expression]` in the following form:
p0.role->exists(p: P | [expression2])
where: `p0` is a parameter of an action *act*; and `role` is a *rolename*, with multiplicity greater than 2 or “*”, of an association between the class of `p0` e another class (**P**). **[expression2]** is another *precond* expression.
	<exists>
	  <parameter name=”p” type ”p1”/>
	  <suchthat>
	    <and>
	      <predicate id=”role”>
	        <parameter id=”p0”/>
	        <parameter id=”p1”/>
	      </predicate>
	    </and>
	  </suchthat>
	</exists>
		`(exists (?p - P)`
		`(and (role ?p0 ?p)`
		`<<expression2 translation>>`
		`)`
		`)`