LTL Translations

BNFs for LTL formulae in Isabelle are as follows:

```
L ::= (alw (L)) | (ev (L)) | (nxt (L)) | ((L) until (L)) | (L C L) | ((L) C (L)) | not (L) | (not (L) | Pred

C ::= aand | or | impl

Pred ::= LabelEq ''string'' | checkInx IOR nat Fun (Value Option) | StateEq (nat Option) | InputEq [Value list] | OutputEq [Value Option list] | InputLength nat | OutputLength nat

IOR ::= ip | op | rg

Value ::= Num nat | Str ''String''

'a Option ::= None | Some a

Fun = ValueEq | ValueGt | ValueLt | ValueGe | ValueLe
```

Normally, Isabelle is pretty lax with bracketing but the LTL package is really strict about it. We can always add more brackets (e.g. we could have (((alw (L)))) but obviously this is silly and we wouldn't do it) but we can't have less brackets unless specified (such as "(L C L)" and "((L) C (L))" are both allowed).

Translations to SAL are as follows:

```
L_t ::= G(Lt) | F(Lt) | X(Lt) | U(Lt, Lt) | Lt Ct Lt | NOT(Lt) | Pred_t

C_t ::= AND | OR | NOT

Value_t ::= NUM nat | STR ''String_t''

Fun_t = value_eq | value_gt | value_lt | value_ge | value_le

String_t = String_String
```

Translating the Preds is a little more complicated as there are a couple of special cases so I'll explain these manually rather than with straight BNFs:

`LabelEq ''string''` goes to `label = String`

`checkInx IOR nat Fun (Value Option)` depends on the value of `IOR`. If it's `ip` then the translation is `gval(Fun_t(I(nat), (Value_t Option)))`. For `op`, the translation is similar but with `O` instead of `I`. If the value of `IOR` is `rg` then we need to index a register so we have `r_nat` in place of `I(nat)` or `O(nat)`.

`StateEq (nat Option)` depends on whether `nat Option` is `None` or `Some n`. If it's `None`, we get `cfstate = NULL_STATE`. If it's `Some n` then we get `cfstate = State_n`.

`InputEq [Value list]` involves recursively building a sequence from the `Value list`. The final translation will be `I = translateValueList`. Translating value lists proceeds as follows:

```
translateValueList([]) = InputSequence ! empty
translateValueList((Num n)#t) = InputSequence ! insert(NUM n, translateValueList(t))
translateValueList((Str s)#t) = InputSequence ! insert(STR String_t, translateValueList(t))
```

`OutputEq [Value Option list]` is similar to `InputEq`. The final translation will be `O = translateValueOptionList` where the translation of the argument proceeds as follows:

```
translateValueOptionList([]) = OutputSequence ! empty
translateValueList((Some (Num n))#t) = OutputSequence ! insert(Some(NUM n), translateValueList(t))
translateValueList((Some (Str s))#t) = OutputSequence ! insert(Some(STR String_t), translateValueList(t))
translateValueList(None#t) = OutputSequence ! insert(None, translateValueList(t))
```

`InputLength nat` goes to `InputSequence ! size?(I)`

`OutputLength nat` goes to `OutputSequence ! size?(O)`