Skolemization for CERES

This semi-formal description of the skolemization algorithm is split into useful (also for other purposes than skolemization itself) portions that can be implemented as operations in CERES' framework. We start with the skolemization operation and list the other ones in order of usage in the main operation

We deal with three different types of occurrences which are all stored in the `Occurrence`-class: formula occurrences, sub-formula occurrences and virtual sub-formula occurrences. The `Occurrence`-class is a quadruple: $\langle$ rule r, sequent-part p, index i, NodePointer t $\rangle$ and denotes the occurrence of the formula or term at position t in the i-th formula of the sequent-part p of the conclusion sequent of r. A formula occurrence is an occurrence where t is the empty pointer and a virtual sub-formula occurrence is a sub-formula occurrence where t is allowed to point outside of the formula. Virtual sub-formula occurrences are useful and necessary to deal with positions inside not-yet-expanded definitions.

1. `LKOpSkolemize` - skolemize an **LK**-derivation

   must only be called for last rule in a proof (`ASSERT( getParent() == NULL )`), does not call itself.

   while ((ρ = `LKOpGetProofNode`(StrongQRuleGoingIntoEndSequent)) != `NULL`) {
   μ := main formula of ρ
   q := `LKOpGetLowestDescendant`(μ)
   f_q := formula occurrence containing q
   F := `LKOpExpandSFOcc`(f_q)
   $\langle (Qx_1), \ldots, (Qx_n) \rangle$ = `F->FOpGetQuantifiers`(`weak`, polarity of f_q in sequent, NodePointer of q)
   let f be a n-ary skolem symbol
   `LKOpWriteSkolemTerm`($f(x_1, \ldots, x_n)$, q)
   }

   Note: The StrongQRuleGoingIntoEndSequent Predicate can be easily implemented in a local way once the derivation is appropriatly marked (which it is, see `Ceres.cpp`).

2. `LKOpGetProofNode` - find a proof node with certain properties

   Input: ProofNodePredicate P

   Output: A (e.g. the leftmost-uppermost) ProofNode fulfilling P or `NULL` if no such ProofNode exists

3. `LKOpGetLowestDescendant` - find lowest descendant of a sub-formula occurrence

Input: sub-formula occurrence μ in a premise of current rule

Output: virtual sub-formula occurrence

If μ has no descendant (i.e. occurs in a cut-formula or in the end-sequent), return μ, else let ν be the descendant of μ and return `LKOpGetLowestDescendant`(ν) - uses `ProofNode::getFDesc`.

4. `LKOpExpandSFOcc` - fully expand a virtual sub-formula occurrence

   Input: virtual sub-formula occcurrence μ in conclusion of current rule

   Output: formula

   returns the formula that would be at μ if all replace-rules on ancestors of μ were fully expanded, throws an exception if inconsistent replace rules are applied to ancestors of μ. This can be implemented by calling `LKOpExpandSFOcc` on all ancestors, checking identity of the formulas on contraction and executing the replacement on replace-rules.

   Example:

$$\dfrac{\dfrac{\dfrac{A \vdash Q(x) \to (Q(y) \vee R(y))}{A \vdash P(x,y)}\, r:r \quad \dfrac{B \vdash Q(x) \to (Q(y) \vee R(y))}{B \vdash P(x,y)}\, r:r}{A \vee B \vdash P(x,y), P(x,y)}\, \vee:l}{A \vee B \vdash P(x,y)}\, c:r$$

   In this example if `LKOpExpandSFOcc` is called with the occurrence $\langle$ lowest rule, consequent, 1st formula, 2.1$\rangle$ it should return $Q(y)$.

5. `FOpGetQuantifiers` - search for quantifiers in a formula

   Input: type parameter t is one of `all, weak, strong`,
   polarity p if t is not `all`,
   optional: NodePointer ω

   Output: list of NodePointers denoting the positions of quantifiers

   Searches for quantifiers of type t (where p is used to determine polarity if t is one of `weak` or `strong`). If ω is given, search for quantifiers only on the path between the root (of the formula) and ω. If ω is not given search for quantifiers in the whole formula.

6. `LKOpWriteSkolemTerm` - skolemize a single quantifier

   Input: skolem term $t = f(x_1, \ldots, x_n)$,
   virtual sub-formula occurrence q of a strong quantifier

   `LKOpWriteSkolemTerm` calls itself for all ancestors of q

   in the conclusion sequent of each rule:
   let y be the variable bound by the quantifier at q,

delete the quantifier at q and
apply the substition $\{y \leftarrow t\}$ to the sub-formula below q (`FOpSubstitute`)

at each weak quantifier rule ρ: apply the substitution of ρ to the skolem term t (to keep it "up-to-date")

If q has no ancestors anymore it is either removed by a weakening rule or it is main formula of a strong quantifier rule ρ. In the first case just stop, in the second case, let α be the eigenvariable of ρ, remove ρ and apply the substition $\{\alpha \leftarrow t\}$ to the proof above ρ (`LKOpSubstitute`).