4.2 Grounding to Parametrized Ground $\mathrm{FO}(\cdot)^{\mathtt{IDP}}$

4.2.1 Phase 1: Simplifying the Syntax

The first phase consists of iterated rewriting of $\mathcal{T}_g$ to obtain a normal form suitable for the effective grounding in phase 2. The phase serves two purposes. First, the resulting theory $\mathcal{T}_g$ will only contain function symbols that are part of . Second, the theory is normalized such that the subsequent grounding will be smaller, faster and introduce less additional symbols (which is well-known to have a detrimental impact on search). Considering the introduction of additional symbols, a rule-of-thumb is that a new symbol is introduced for each "context switch" when descending through the parse-tree, such as a conjunction nested below a disjunction or a product below an addition.

We first introduce rules for the first property, the only one required for correctness of the grounding algorithm. Afterwards, we present the rules to normalize the theory further.

Normalization

After unnesting and graphing, the following rules are applied to bring the theory in a form with a smaller grounding with less additional symbols. This is achieved among others by reducing the number of context switches and pushing quantifications lower in the parse-tree.

- PUSH-NEGATIONS applies the standard equivalence-preserving transformations to push negations down.[4]

$$\neg \bigwedge_{i\in S} \varphi_i \rightarrowtail \bigvee_{i\in S} \neg\varphi_i \qquad \neg \bigvee_{i\in S} \varphi_i \rightarrowtail \bigwedge_{i\in S} \neg\varphi_i$$

$$\neg\forall \overline{x} : \varphi \rightarrowtail \exists \overline{x} : \neg\varphi \qquad \neg\exists \overline{x} : \varphi \rightarrowtail \forall \overline{x} : \neg\varphi$$

$$\neg\neg\varphi \rightarrowtail \varphi \qquad \neg(t \sim t') \rightarrowtail t \not\sim t'$$

- PUSH-QUANTIFIERS pushes down quantifiers to where they are relevant. The following rule pushes a variable x_j down if it only occurs in some subformulas of a disjunction:

$$\forall x_1 \dots x_n : \bigvee_{\varphi\in S} \varphi \rightarrowtail \forall x_1 \dots x_{j-1}\, x_{j+1} \dots x_n : \Big(\bigvee_{\varphi\in S_1} \vee (\forall x_j : \bigvee_{\varphi\in S_2})\Big)$$

[4] Given a comparison $\sim$, we use $\not\sim$ to denote its negation.

with S_1 and S_2 the partition of S such that x_j does not occur in formulas in S_1 and occurs in all formulas in S_2. A similar rule is applied for all other combinations of universal/existential quantifications with conjunctive/disjunctive subformulas. For definitions, the following rule is also applied, which moves variables from the head to the body.

$$\forall \overline{x} : head \leftarrow \varphi \rightarrowtail \forall x_1 \dots x_{j-1}\, x_{j+1} \dots x_n : head \leftarrow \exists x_j : \varphi$$

Pushing x_j's quantification down is only allowed if the interpretation of its type is not empty in $\mathcal{I}_{in}$. In that case, other simplifications are available (see below).

- FLATTEN combines subsequent identical operators into one:

$$\bigwedge_{\varphi \in S} \varphi \rightarrowtail \bigwedge_{\varphi \in S - \psi \cup S'} \varphi \qquad \textbf{if } \psi \in S \text{ and } \psi = \textstyle\bigwedge_{\varphi \in S'} \varphi$$

$$\bigvee_{\varphi \in S} \varphi \rightarrowtail \bigvee_{\varphi \in S - \psi \cup S'} \varphi \qquad \textbf{if } \psi \in S \text{ and } \psi = \textstyle\bigvee_{\varphi \in S'} \varphi$$

$$\forall \overline{x} : \forall \overline{y} : \varphi \rightarrowtail \forall \overline{x} :: \overline{y} : \varphi$$

$$\exists \overline{x} : \exists \overline{y} : \varphi \rightarrowtail \exists \overline{x} :: \overline{y} : \varphi$$

$$agg(agg(S_1), agg(S_2)) \rightarrowtail agg(S_1 \cup S_2)$$

Most of the transformations are implemented in IDP as an operation on the parse-tree that takes as input a term, formula or theory and visits the input in a top-down, depth-first fashion, rewriting it on-the-go to satisfy the appropriate post conditions.

4.2.2 Phase 2: Grounding

The aim of the second phase is to produce the effective grounding, in an appropriate normal form so that relevant fragments can easily be supported by solvers. This normal form is the so-called Extended Conjunctive Normal Form (ECNF), defined as follows.

Definition 4.2.1 (Extended CNF)**.** An FO(·)$^{\text{IDP}}$-theory is in ECNF if all its sentences are either disjunctions of domain atoms $L_1 \vee \dots \vee L_n$ or definitions where rules are of one the following forms (with all L_i's

domain literals and all e_i's constants or domain elements):

$$P(\overline{e}) \leftarrow L_1 \wedge \ldots \wedge L_n \qquad P(\overline{e}) \leftarrow L_1 \vee \ldots \vee L_n$$

$$P(\overline{e}) \leftarrow Q(\overline{e}') \qquad P(\overline{e}) \leftarrow f(\overline{e}) \sim e_0$$

$$P(\overline{e}) \leftarrow agg(\{L_1 : e_1\} \cup \cdots \cup \{L_n : e_n\}) \sim e_0$$

Any FO$(\cdot)^{\text{IDP}}$-theory over a finite structure $\mathcal{I}$ can be reduced to a $\{\Sigma, \mathcal{I}\}$-equivalent-theory in ECNF.

From now on, the domains of variables are made explicit in all expressions, written as $\forall \overline{x} \in \overline{D} : \varphi$ or $\{\overline{x} \in \overline{D} : \varphi : t\}$. Initially, $\overline{D}$ is $\tau_1^{\mathcal{I}_{in}} \times \cdots \times \tau_n^{\mathcal{I}_{in}}$, where τ_i is the type of x_i (recall that $\mathcal{I}_{in}$ interprets all types).

The second phase then consists of applying the following set of rewrite rules to effectively instantiate variables and introduce new symbols to obtain the above normal form. The phase terminates when no more rules are applicable. For this phase, we introduce a term **undef** which represents a non-denoting term; it is only used during grounding and will not occur in the resulting ground theory.

– INSTANTIATE, for some $\overline{d} \in \overline{D}$:

$$\forall \overline{x} \in \overline{D} : \varphi \rightarrowtail \varphi[\overline{x}/\overline{d}] \wedge \forall \overline{x} \in \overline{D} - \overline{d} : \varphi$$

$$\exists \overline{x} \in \overline{D} : \varphi \rightarrowtail \varphi[\overline{x}/\overline{d}] \vee \exists \overline{x} \in \overline{D} - \overline{d} : \varphi$$

$$\forall \overline{x} \in \overline{D} : head \leftarrow \varphi \rightarrowtail head[\overline{x}/\overline{d}] \leftarrow \varphi[\overline{x}/\overline{d}], \quad \forall \overline{x} \in \overline{D} - \overline{d} : head \leftarrow \varphi$$

$$\{\overline{x} \in \overline{D} : \varphi : t\} \rightarrowtail \{\varphi[\overline{x}/\overline{d}] : t[\overline{x}/\overline{d}]\} \cup \{\overline{x} \in \overline{D} - \overline{d} : \varphi : t\}$$

– INTRODUCE TSEITIN $\varphi \rightarrowtail T_\varphi$, where φ is an occurrence of a formula without free variables in $\mathcal{T}_g$ and T_φ is a new propositional symbol. In addition, a rule is added that defines T_φ: if φ occurs in a definition Δ, the rule $T_\varphi \leftarrow \varphi$ is added to Δ, otherwise, the singleton definition $\{T_\varphi \leftarrow \varphi\}$ is added to $\mathcal{T}_g$.
The rule is not applied if φ is a literal, a sentence or a rule body. Recall that a sufficient condition to preserve model equivalence is to not introduce Tseitins under negation, which is guaranteed by the rule PUSH-NEGATIONS applied in the previous phase and by restricting the nesting of aggregate terms over inductively defined symbols.

– INTRODUCE TERM $t \mapsto c_t$, where t is an occurrence of a term without free variables in $\mathcal{T}_g$ and c_t is a newly introduced constant $c_t[\mapsto type(t)]$. The sentence $t = c_t$ is added to $\mathcal{T}_g$. Informally, the idea is to reduce the nesting depth of terms that are supported by the search algorithm to obtain ECNF expressions. The rule is not applied if t is a domain element, a variable or a constant, or if t occurs as $t \sim c'$ or $c' \sim t$.

– EVALUATE domain terms and atoms in the structure

$$f(\overline{d}) \mapsto d' \qquad \textbf{if } f_{ct}^{\mathcal{I}_{in}}(\overline{d}) = \{d'\}$$

$$f(\overline{d}) \mapsto \textbf{undef} \qquad \textbf{if } f_{pt}^{\mathcal{I}_{in}}(\overline{d}) = \emptyset \textbf{ or } d_i \notin type(arg_i(f))^{\mathcal{I}_{in}}$$

$$P(\overline{d}) \mapsto \top \qquad \textbf{if } P(\overline{d})^{\mathcal{I}_{in}} = \textbf{t}$$

$$P(\overline{d}) \mapsto \bot \qquad \textbf{if } P(\overline{d})^{\mathcal{I}_{in}} = \textbf{f} \textbf{ or } d_i \notin type(arg_i(P))^{\mathcal{I}_{in}}$$

The rules for a symbol s are not applied in definitions that define s, as loops through the definition might be lost (which leads to inconsistencies). More specifically, if the definition is monotone, replacements with $\bot$ are allowed.

Note that application of built-in symbols such as aggregates, arithmetic and constructed types is covered by EVALUATE, as they are interpreted in $\mathcal{I}_{in}$. We add two more interesting rules related to function terms under equality (and, similarly, in inequality):

$$f(\overline{d}) = d \mapsto \top \qquad \textbf{if } d \in f_{ct}^{\mathcal{I}_{in}}(\overline{d})$$

$$f(\overline{d}) = d \mapsto \bot \qquad \textbf{if } d \in f_{cf}^{\mathcal{I}_{in}}(\overline{d})$$

For aggregates, a similar rule is possible, which allows us to only partially ground aggregate sets. It follows from the fact that all aggregates are composable. For any aggregate function, we can straightforwardly derive minimum and maximum bounds given a set S (even a quantified one). For example for an expression $sum(\{\overline{x} \in \overline{D} : \varphi : t\})$, the minimum bound min_{bound} is $min[\mapsto type(t)]^{\mathcal{I}} \times |\overline{D}|$ and the maximum bound max_{bound} is $max[\mapsto type(t)]^{\mathcal{I}} \times |\overline{D}|$. If we then have an atom $agg(S \cup \{\top : d\} \cup S') \sim d'$ in $\mathcal{T}_g$, simplifications are possible in various cases (not enumerated here). One example is in case of a sum aggregate with $>$ comparison operator: if $d + min(min_{bound}, 0) > d'$, then the atom evaluates to true.

– SPLIT *conjunctive sentences:* $\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrowtail \varphi_1, \ldots, \varphi_n$.

– SIMPLIFY
Basic simplifications

$$
\begin{array}{lll}
\neg\top \rightarrowtail \bot & & \\
\neg\bot \rightarrowtail \top & & \\
\varphi_1 \vee \ldots \vee \varphi_n \rightarrowtail \top & & \textbf{if } \varphi_j = \top \\
\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrowtail \bot & & \textbf{if } \varphi_j = \bot \\
\varphi_1 \vee \ldots \vee \varphi_n \rightarrowtail \varphi_1 \vee \ldots \vee \varphi_{j-1} \vee \varphi_{j+1} \vee \ldots \vee \varphi_n & & \textbf{if } \varphi_j = \bot \\
\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrowtail \varphi_1 \wedge \ldots \wedge \varphi_{j-1} \wedge \varphi_{j+1} \wedge \ldots \wedge \varphi_n & & \textbf{if } \varphi_j = \top \\
a \leftarrow \top,\ a \leftarrow \varphi \rightarrowtail a \leftarrow \top & & \\
a \leftarrow \bot,\ a \leftarrow \varphi \rightarrowtail a \leftarrow \varphi & &
\end{array}
$$

Quantification simplifications

$$
\begin{array}{lll}
\forall \overline{x} \in \overline{D} : \varphi & \rightarrowtail \top & \textbf{if } \overline{D} = \emptyset \text{ or } \varphi = \top \\
\exists \overline{x} \in \overline{D} : \varphi & \rightarrowtail \bot & \textbf{if } \overline{D} = \emptyset \text{ or } \varphi = \bot \\
\{\overline{x} \in \overline{D} : \varphi : t\} & \rightarrowtail \{\bot : \textbf{undef}\} & \textbf{if } \overline{D} = \emptyset \text{ or } \varphi = \bot
\end{array}
$$

Partial function simplifications

$$
\begin{array}{lll}
f(t_1, \ldots, t_n) \rightarrowtail \textbf{undef} & & \textbf{if } t_j = \textbf{undef} \\
P(t_1, \ldots, t_n) \rightarrowtail \bot & & \textbf{if } t_j = \textbf{undef}
\end{array}
$$

Aggregate simplifications

$$
agg(S \cup \{\bot : t\}) \rightarrowtail agg(S)
$$

$$
agg(S \cup \{\varphi : d\} \cup \{\varphi : d'\}) \rightarrowtail agg(S \cup \{\varphi : agg(\{\top : d\} \cup \{\top : d'\})^{\mathcal{I}_{in}}\})
$$

After application of the above rewrite rules, we obtain a theory in ECNF.