Motivation

1 Algorithm: WhatDoIDo\((n, m)\)

Input : Two positive integers \(n, m\).

Output: The number contained in \(n\).

2 while \((m > 0)\) do

3 \hspace{1em} m = m -1 ;

4 \hspace{1em} n = n + 1;

5 end

6 return \(n\);
In First-Order Logic Modulo LIA

2. \( \forall n, m. \quad (m > 0, R(2, n, m) \rightarrow R(3, n, m)) \)

2. \( \forall n, m. \quad (m = 0, R(2, n, m) \rightarrow R(6, n, m)) \)

3. \( \forall n, m, m'. \quad (m' = m - 1, R(3, n, m) \rightarrow R(4, n, m')) \)

4. \( \forall n, m, n'. \quad (n' = n + 1, R(4, n, m) \rightarrow R(5, n', m)) \)

5. \( \forall n, m. \quad (R(5, n, m) \rightarrow R(2, n, m)) \)
In First-Order Logic Modulo LIA

2  \( \forall n, m. \)  \((m > 0, R(2, n, m) \rightarrow R(3, n, m))\)

2  \( \forall n, m. \)  \((m = 0, R(2, n, m) \rightarrow R(6, n, m))\)

3  \( \forall n, m, m'. \)  \((m' = m - 1, R(3, n, m) \rightarrow R(4, n, m'))\)

4  \( \forall n, m, n'. \)  \((n' = n + 1, R(4, n, m) \rightarrow R(5, n', m))\)

5  \( \forall n, m. \)  \((R(5, n, m) \rightarrow R(2, n, m))\)

\( \forall n, m . \)  \((R(2, n, m) \rightarrow R(6, n + m, 0))\)
2-Counter Machines (Minsky 1967)

The memory of the machine are two integer counters $k_1, k_2$, where the integers are not limited in size, resulting in the name. The counters may be initialized at the beginning with arbitrary positive values.

A program consists of a finite number of programming lines, each coming with a unique and consecutive line number and containing exactly one instruction. The available instructions are:

- $\text{inc}(k_i)$  increment counter $k_i$ and goto the next line,
- $\text{td}(k_i, n)$  if $k_i > 0$ then decrement $k_i$ and goto the next line, otherwise goto line $n$ and leave counters unchanged,
- $\text{goto } n$  goto line $n$,
- $\text{halt}$  halt the computation.
Example: WhatDoIDo

2 \text{td}(k_2, 6)
4 \text{inc}(k_1)
5 \text{goto} 2
6 \text{halt}
8.10.1 Theorem (2-Counter Machine Halting Problem)

The halting problem for 2-counter machines is undecidable (Minsky 1967).

Proof.

(Idea) By a reduction to the halting problem for Turing machines.

8.10.2 Proposition (FOL(LIA) Undecidability with a Single Ternary Predicate)

Unsatisfiability of a FOL(LIA) clause set with a single ternary predicate is undecidable.
FOL(LIA) Decidable for Binary or Monadic Predicates?

No: translate 2-counter machine halting problem to FOL(LIA) with a single monadic predicate.

Idea: translate state \((i, n, m)\) where the program is at line \(i\) with respective counter values \(n, m\) by the integer \(2^n \cdot 3^m \cdot p_i\) where \(p_i\) is the \(i^{th}\) prime number following 3
**Example: WhatDoIDo**

1. \( \text{td}(k_2, 4) \)
2. \( \text{inc}(k_1) \)
3. goto 1
4. halt
Example: WhatDoIDo

1  td(k_2, 4)
2  inc(k_1)
3  goto 1
4  halt

5y = x, 3y' = y, x' = 7y', S(x) \rightarrow S(x')
5y = x, 3y' + 1 = y, x' = 13y', S(x) \rightarrow S(x')
5y = x, 3y' + 2 = y, x' = 13y', S(x) \rightarrow S(x')
7y = x, x' = 2y, x'' = 11x', S(x) \rightarrow S(x'')
11y = x, x' = 5y, S(x) \rightarrow S(x')
13y = x, S(x) \rightarrow
8.10.3 Proposition (FOL(LIA) Undecidability with a Single Monadic Predicate)

Unsatisfiability of a FOL(LIA) clause set with a single monadic predicate is undecidable (Downey 1972).
8.2.1 Definition (Hierarchic Theory and Specification)

Let $\mathcal{T}^B = (\Sigma^B, C^B)$ be a many-sorted theory, called the background theory and $\Sigma^B$ the background signature. Let $\Sigma^F$ be a many sorted signature with $\Omega^B \cap \Omega^F = \emptyset$, $S^B \subset S^F$, called the foreground signature or free signature. Let $\Sigma^H = (S^B \cup S^F, \Omega^B \cup \Omega^F)$ be the union signature and $N$ be a set of clauses over $\Sigma^H$, and $\mathcal{T}^H = (\Sigma^H, N)$ called a hierarchic theory. A pair $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$ is called a hierarchic specification.
I abbreviate $\models_{TB} \phi \ (\models_{TH} \phi)$ with $\models_B \phi \ (\models_H \phi)$, meaning that $\phi$ is valid in the respective theory, see Definition 3.17.1.

Terms, atoms, literals build over $\Sigma^B$ are called *pure background terms*, *pure background atoms*, and *pure background literals*, respectively. Non-variable terms, atoms, literals build over $\Sigma^F$ are called *free terms*, *free atoms*, *free literals*. A variable of sort $S \in (S^F \setminus S^B)$ is also called a *free variable* and a *free term*. Any term of some sort $S \in S^B$ built out of $\Sigma^H$ is called a *background term*.

A substitution $\sigma$ is called *simple* if $x_S \sigma \in T_S(\Sigma^B, \mathcal{A})$ for all $S \in S^B$. 
8.2.2 Example (Classes of Terms)

Let $\mathcal{T}^B$ be linear rational arithmetic and $\Sigma^F = (\{S, LA\}, \{g, a\})$ where $a: S$ and $g: LA \to LA$. Then the terms $x_{LA} + 3$ and $g(x_{LA})$ are all of sort $LA$, but $x_{LA} + 3$ is a pure background term whereas $g(x_{LA})$ is a free term and an unpure background term. So the substitution $\sigma = \{y_{LA} \mapsto x_{LA} + 3\}$ is simple while $\sigma = \{y_{LA} \mapsto g(x_{LA})\}$ is not.
8.2.3 Definition (Hierarchic Algebras)

Given a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, $\mathcal{T}^B = (\Sigma^B, C^B)$, $\mathcal{T}^H = (\Sigma^H, N)$, a $\Sigma^H$-algebra $\mathcal{A}$ is called hierarchic if $\mathcal{A}|_{\Sigma^B} \in C^B$. A hierarchic algebra $\mathcal{A}$ is called a model of a hierarchic specification $\mathcal{H}$, if $\mathcal{A} \models N$. 
8.2.4 Definition (Abstracted Term, Atom, Literal, Clause)

A term $t$ is called *abstracted* with respect to a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, if $t \in T_S(\Sigma^B, \mathcal{X})$ or $t \in T_T(\Sigma^F, \mathcal{X})$ for some $S \in S^B$, $T \in S^B \cup S^F$. An equational atom $t \approx s$ is called *abstracted* if $t$ and $s$ are abstracted and both pure or both unpure, accordingly for literals. A clause is called *abstracted* of all its literals are abstracted.
Abstraction  \[ N \uplus \{ C \lor E[t]_p[s]_q \} \implies \text{ABSTR} \]
\[ N \uplus \{ C \lor x_s \not\approx s \lor E[x_S]_q \} \]
provided \( t, s \) are non-variable terms, \( q \not\approx p \), \( \text{sort}(s) = S \), and
either \( \text{top}(t) \in \Sigma^F \) and \( \text{top}(s) \in \Sigma^B \) or \( \text{top}(t) \in \Sigma^B \) and \( \text{top}(s) \in \Sigma^F \)
8.2.5 Proposition (Properties of the Abstraction)

Given a finite clause set $N$ out of a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, $\Rightarrow_{\text{ABSTR}}$ terminates on $N$ and preserves satisfiability. For any clause $C \in (N \Downarrow_{\text{ABSTR}})$ and any literal $E \in C$, $E$ does not both contain a function symbol from $\Sigma^B$ and a function symbol from $\Sigma^F$. 
From now on I assume fully abstracted clauses $C$, i.e., for all atoms $s \equiv t$ occurring in $C$, either $s, t \in T(\Sigma^B, \mathcal{X})$ or $s, t \in T(\Sigma^F, \mathcal{X})$. This justifies the notation of clauses $\Lambda \parallel C$ where all pure background literals are in $\Lambda$ and belong to $\text{FOL}(\Sigma^B, \mathcal{X})$ and all literals in $C$ belong to $\text{FOL}(\Sigma^F, \mathcal{X})$.

The literals in $\Lambda$ form a conjunction and the literals in $C$ a disjunction and the overall clause the implication $\Lambda \rightarrow C$. For a clause $\Lambda \parallel C$ the background theory part $\Lambda$ is called the constraint and $C$ the free part of the clause.
8.2.6 Example (Abstracted Clause)

Continuing Example 8.2.2, the unabstracted clause

\[ g(x) \leq 1 + y \lor g(g(1)) \approx 2 \]

corresponds to the abstracted clause

\[ z \not\approx g(x) \lor z \leq 1 + y \lor u \not\approx 2 \lor v \not\approx 1 \lor g(g(v)) \approx u \]

that is written

\[ z > 1 + y \land u \approx 2 \land v \approx 1 \lor z \not\approx g(x) \lor g(g(v)) \approx u \]
SUP(T) on Abstracted Clauses

As usual the calculus is presented with respect to a reduction ordering $\prec$, total on ground terms. For the SUP(T) calculus I assume that any pure base term is strictly smaller than any term containing a function symbol from $\Sigma^F$. This justifies the below ordering conditions with respect to the constraint notation of clauses and can, e.g., be obtained by an LPO where all symbols from $\Sigma^B$ are smaller in the precedence than the symbols from $\Sigma^F$. 
Superposition Right
\(( N \cup \{ \Lambda \mid D \lor t \approx t', \Gamma \mid C \lor s[u] \approx s' \} ) \Rightarrow_{\text{SUPT}} ( N \cup \{ \Lambda \mid D \lor t \approx t', \Gamma \mid C \lor s[u] \approx s' \} \cup \{ (\Lambda, \Gamma \mid D \lor C \lor s[t'] \approx s')_{\sigma} \} ) \)

where \( \sigma \) is the mgu of \( t, u \), \( \sigma \) is simple, \( u \) is not a variable
\( t_{\sigma} \nless t'_{\sigma}, s_{\sigma} \nless s'_{\sigma}, (t \approx t')_{\sigma} \) strictly maximal in \((D \lor t \approx t')_{\sigma}\),
nothing selected and \((s \approx s')_{\sigma} \) maximal in \((C \lor s \approx s')_{\sigma} \) and
nothing selected

Superposition Left
\(( N \cup \{ \Lambda \mid D \lor t \approx t', \Gamma \mid C \lor s[u] \not\approx s' \} ) \Rightarrow_{\text{SUPT}} ( N \cup \{ \Lambda \mid D \lor t \approx t', \Gamma \mid C \lor s[u] \not\approx s' \} \cup \{ (\Lambda, \Gamma \mid D \lor C \lor s[t'] \not\approx s')_{\sigma} \} ) \)

where \( \sigma \) is the mgu of \( t, u \), \( \sigma \) is simple, \( u \) is not a variable
\( t_{\sigma} \nless t'_{\sigma}, s_{\sigma} \nless s'_{\sigma}, (t \approx t')_{\sigma} \) strictly maximal in \((D \lor t \approx t')_{\sigma}\), nothing
selected and \((s \not\approx s')_{\sigma} \) maximal in \((C \lor s \not\approx s')_{\sigma} \) or selected
Equality Resolution

\[ (N \cup \{ \Gamma \mid C \lor s \not\approx s' \}) \]
\[ \Rightarrow_{\text{SUPT}} (N \cup \{ \Gamma \mid C \lor s \not\approx s' \} \cup \{(\Gamma \mid C)\sigma\}) \]
where \(\sigma\) is the mgu of \(s, s', \sigma\) is simple, \((s \not\approx s')\sigma\) maximal in \((C \lor s \not\approx s')\sigma\) or selected

Equality Factoring

\[ (N \cup \{ \Gamma \mid C \lor s' \approx t' \lor s \approx t \}) \]
\[ \Rightarrow_{\text{SUPT}} (N \cup \{ \Gamma \mid C \lor s' \approx t' \lor s \approx t \} \cup \{(\Gamma \mid C \lor t \not\approx t' \lor s \approx t')\sigma\}) \]
where \(\sigma\) is the mgu of \(s, s', \sigma\) is simple, \(s'\sigma \not\leq t'\sigma, s\sigma \not\leq t\sigma, (s \approx t)\sigma\) maximal in \((C \lor s' \approx t' \lor s \approx t)\sigma\) and nothing selected

Constraint Refutation

\[ (N \cup \{ \Gamma_1 \mid \bot, \ldots, \Gamma_n \mid \bot \}) \]
\[ \Rightarrow_{\text{SUPT}} (N \cup \{ \Gamma_1 \mid \bot, \ldots, \Gamma_n \mid \bot \} \cup \{\bot\}) \]
where \(\Gamma_1 \mid \bot \land \ldots \land \Gamma_n \mid \bot \models_B \bot\)
8.3.1 Definition (Sufficient Completeness)

A hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$ is *sufficiently complete* with respect to simple ground instances if for all unpure ground terms $t$ of a background sort, there exists a pure ground term $t'$ of the same sort such that $\mathcal{A} \models t \approx t'$ for all $\mathcal{A}$ algebras with $\mathcal{A} \models \text{sgr}(N) \cup \text{grd}(\mathcal{T}^B)$ where $\text{grd}(\mathcal{T}^B)$ is the set of all ground formulas $\phi$ over $\Sigma^B$ with $\models_B \phi$. 
8.3.2 Definition (SUP(T) Abstract Redundancy)

A clause $\Gamma \parallel C$ is \textit{redundant} with respect to a clause set $N$ if for all simple ground instances $(\Gamma \parallel C_\sigma)$ there are clauses $\{\Lambda_1 \parallel C_1, \ldots, \Lambda_n \parallel C_n\} \subseteq N$ with simple ground instances $(\Lambda_1 \parallel C_1)_{\tau_1}, \ldots, (\Lambda_n \parallel C_n)_{\tau_n}$ such that $(\Lambda_i \parallel C_i)_{\tau_i} \prec (\Gamma \parallel C)_\sigma$ for all $i$ and $(\Lambda_1 \parallel C_1)_{\tau_1}, \ldots, (\Lambda_n \parallel C_n)_{\tau_n} \models_B (\Gamma \parallel C)_\sigma$. 
8.3.3 Theorem (SUP(T) Completeness)

Let $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$ be sufficiently complete and $\mathcal{T}^B$ be compact and term-generated. Then $N$ is unsatisfiable with respect to hierarchic algebras of $\mathcal{H}$ iff $N \Rightarrow_{\text{SUP}}^* N' \cup \{\bot\}$.