Motivation

```
Algorithm: WhatDoIDo(n, m)
Input  : Two positive integers n, m.
Output: The number contained in n.
while (m > 0) do
|   m = m -1 ;
|   n = n + 1;
end
return n;
```

In First-Order Logic Modulo LIA

$$
\begin{array}{lllll}
2 & \forall n, m. & (m > 0, R(2,n,m) & \rightarrow R(3,n,m)) \\
2 & \forall n, m. & (m = 0, R(2,n,m) & \rightarrow R(6,n,m)) \\
3 & \forall n, m, m'. & (m' = m - 1, R(3,n,m) & \rightarrow R(4,n,m')) \\
4 & \forall n, m, n'. & (n' = n + 1, R(4,n,m) & \rightarrow R(5,n',m)) \\
5 & \forall n, m. & (R(5,n,m) & \rightarrow R(2,n,m))
\end{array}
$$

In First-Order Logic Modulo LIA

$$
\begin{array}{llrl}
2 & \forall n, m. & (m > 0, R(2, n, m) & \rightarrow R(3, n, m)) \\
2 & \forall n, m. & (m = 0, R(2, n, m) & \rightarrow R(6, n, m)) \\
3 & \forall n, m, m'. & (m' = m - 1, R(3, n, m) & \rightarrow R(4, n, m')) \\
4 & \forall n, m, n'. & (n' = n + 1, R(4, n, m) & \rightarrow R(5, n', m)) \\
5 & \forall n, m. & (R(5, n, m) & \rightarrow R(2, n, m))
\end{array}
$$

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

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:

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

Example: WhatDoIDo

2 $\mathrm{td}(k_2, 6)$
4 $\mathrm{inc}(k_1)$
5 goto 2
6 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 td(k_2, 4)
2 inc(k_1)
3 goto 1
4 halt

Example: WhatDoIDo

1 $\text{td}(k_2, 4)$
2 $\text{inc}(k_1)$
3 $\text{goto}\,1$
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).

Syntax and Semantics

8.2.1 Definition (Hierarchic Theory and Specification)

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

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

A substitution σ is called *simple* if $x_S\sigma \in T_S(\Sigma^B, \mathcal{X})$ for all $S \in \mathcal{S}^B$.

8.2.2 Example (Classes of Terms)

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

8.2.3 Definition (Hierarchic Algebras)

Given a hierarchic specification $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, $\mathcal{T}^B = (\Sigma^B, \mathcal{C}^B)$, $\mathcal{T}^H = (\Sigma^H, N)$, a Σ^H-algebra $\mathcal{A}$ is called *hierarchic* if $\mathcal{A}|_{\Sigma^B} \in \mathcal{C}^B$. 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 $S \in \mathcal{S}^B$, $T \in \mathcal{S}^B \cup \mathcal{S}^F$. 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 $N \uplus \{C \vee E[t]_p[s]_q\} \Rightarrow_{\text{ABSTR}}$
$N \cup \{C \vee x_s \not\approx s \vee E[x_S]_q\}$

provided t, s are non-variable terms, $q \not< p$, $\mathrm{sort}(s) = S$, and either $\mathrm{top}(t) \in \Sigma^F$ and $\mathrm{top}(s) \in \Sigma^B$ or $\mathrm{top}(t) \in \Sigma^B$ and $\mathrm{top}(s) \in \Sigma^F$

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 Σ^B and a function symbol from Σ^F.

From now on I assume fully abstracted clauses C, i.e., for all atoms $s \approx t$ 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 Λ and belong to $\mathrm{FOL}(\Sigma^B, \mathcal{X})$ and all literals in C belong to $\mathrm{FOL}(\Sigma^F, \mathcal{X})$.

The literals in Λ 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 Λ 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

$$g(x) \leq 1 + y \vee g(g(1)) \approx 2$$

corresponds to the abstracted clause

$$z \not\approx g(x) \vee z \leq 1 + y \vee u \not\approx 2 \vee v \not\approx 1 \vee g(g(v)) \approx u$$

that is written

$$z > 1 + y \wedge u \approx 2 \wedge v \approx 1 \parallel z \not\approx g(x) \vee g(g(v)) \approx u$$

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 Σ^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 Σ^B are smaller in the precedence than the symbols from Σ^F.

Superposition Right

$(N \uplus \{\Lambda \parallel D \vee t \approx t', \Gamma \parallel C \vee s[u] \approx s'\}) \Rightarrow_{\text{SUPT}} (N \cup \{\Lambda \parallel D \vee t \approx t', \Gamma \parallel C \vee s[u] \approx s'\} \cup \{(\Lambda, \Gamma \parallel D \vee C \vee s[t'] \approx s')\sigma\})$

where σ is the mgu of t, u, σ is simple, u is not a variable $t\sigma \not\preceq t'\sigma$, $s\sigma \not\preceq s'\sigma$, $(t \approx t')\sigma$ strictly maximal in $(D \vee t \approx t')\sigma$, nothing selected and $(s \approx s')\sigma$ maximal in $(C \vee s \approx s')\sigma$ and nothing selected

Superposition Left

$(N \uplus \{\Lambda \parallel D \vee t \approx t', \Gamma \parallel C \vee s[u] \not\approx s'\}) \Rightarrow_{\text{SUPT}} (N \cup \{\Lambda \parallel D \vee t \approx t', \Gamma \parallel C \vee s[u] \not\approx s'\} \cup \{(\Lambda, \Gamma \parallel D \vee C \vee s[t'] \not\approx s')\sigma\})$

where σ is the mgu of t, u, σ is simple, u is not a variable $t\sigma \not\preceq t'\sigma$, $s\sigma \not\preceq s'\sigma$, $(t \approx t')\sigma$ strictly maximal in $(D \vee t \approx t')\sigma$, nothing selected and $(s \not\approx s')\sigma$ maximal in $(C \vee s \not\approx s')\sigma$ or selected

Equality Resolution $(N \uplus \{\Gamma \parallel C \vee s \not\approx s'\})$
$\Rightarrow_{\text{SUPT}}$ $(N \cup \{\Gamma \parallel C \vee s \not\approx s'\} \cup \{(\Gamma \parallel C)\sigma\})$
where σ is the mgu of s, s', σ is simple, $(s \not\approx s')\sigma$ maximal in $(C \vee s \not\approx s')\sigma$ or selected

Equality Factoring $(N \uplus \{\Gamma \parallel C \vee s' \approx t' \vee s \approx t\})$
$\Rightarrow_{\text{SUPT}}$
$(N \cup \{\Gamma \parallel C \vee s' \approx t' \vee s \approx t\} \cup \{(\Gamma \parallel C \vee t \not\approx t' \vee s \approx t')\sigma\})$
where σ is the mgu of s, s', σ is simple, $s'\sigma \not\preceq t'\sigma$, $s\sigma \not\preceq t\sigma$, $(s \approx t)\sigma$ maximal in $(C \vee s' \approx t' \vee s \approx t)\sigma$ and nothing selected

Constraint Refutation $(N \uplus \{\Gamma_1 \parallel \bot, \ldots, \Gamma_n \parallel \bot\})$
$\Rightarrow_{\text{SUPT}}$ $(N \cup \{\Gamma_1 \parallel \bot, \ldots, \Gamma_n \parallel \bot\} \cup \{\bot\})$
where $\Gamma_1 \parallel \bot \wedge \ldots \wedge \Gamma_n \parallel \bot \models_B \bot$

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 $\mathcal{A} \models \text{sgi}(N) \cup \text{grd}(\mathcal{T}^B)$ where $\text{grd}(\mathcal{T}^B)$ is the set of all ground formulas ϕ over Σ^B with $\models_B \phi$.

8.3.2 Definition (SUP(T) Abstract Redundancy)

A clause $\Gamma \parallel C$ is *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 $N \Rightarrow^*_{\text{SUPT}} N' \cup \{\bot\}$.