max planck institut
informatik

Automated Reasoning

**Martin Bromberger, Sibylle Möhle,
Simon Schwarz, Christoph Weidenbach**

Max Planck Institute for Informatics

January 25, 2023

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

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

In First-Order Logic Modulo LIA

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

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

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:

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

Example: WhatDoIDo

2 $\mathsf{td}(k_2, 6)$

4 $\mathsf{inc}(k_1)$

5 $\mathsf{goto}\,2$

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

Example: WhatDoIDo

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

$$5y = x, 3y' = y, x' = 7y', S(x) \to S(x')$$
$$5y = x, 3y' + 1 = y, x' = 13y', S(x) \to S(x')$$
$$5y = x, 3y' + 2 = y, x' = 13y', S(x) \to S(x')$$
$$7y = x, x' = 2y, x'' = 11x', S(x) \to S(x'')$$
$$11y = x, x' = 5y, S(x) \to S(x')$$
$$13y = x, S(x) \to$$

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*.

A constant $c \in \Omega^{\mathcal{B}}$ is called a *domain constant* if $c^{\mathcal{A}} \neq d^{\mathcal{A}}$ for all $\mathcal{A} \in \mathcal{C}^{\mathcal{B}}$ and for all $d \in \Omega^{\mathcal{B}}$ with $d \neq c$.

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.

All terms, atoms, with a top-symbol from $\Omega^{\mathcal{B}}$ or $\Pi^{\mathcal{B}}$, respectively, are called *background terms*, *background atoms*, respectively. A background atom or its negation is a *background literal*.

All terms, atoms, with a top-symbol from $\Omega^{\mathcal{F}}$ or $\Pi^{\mathcal{F}}$, respectively, are called *foreground terms*, *foreground atoms*, respectively. A foreground atom or its negation is a *foreground literal*.

Given a set (sequence) of $\mathcal{H}$ literals, the function bgd returns the set (sequence) of background literals and the function fgd the respective set (sequence) of foreground literals.

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

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 an impure foreground 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.

$g(x_{LA}+3)$ an impure foreground term

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.

As usual, $\mathcal{A}^{\mathcal{H}}|_{\Sigma^{\mathcal{B}}}$ is obtained from a $\mathcal{A}^{\mathcal{H}}$-algebra by removing all carrier sets $S^{\mathcal{A}}$ for all $S \in (\mathcal{S}^{\mathcal{F}} \setminus \mathcal{S}^{\mathcal{B}})$, all functions from $\Omega^{\mathcal{F}}$ and all predicates from $\Pi^{\mathcal{F}}$. We write $\models_{\mathcal{H}}$ for the entailment relation with respect to hierarchic algebras and formulas from $\Sigma^{\mathcal{H}}$ and $\models_{\mathcal{B}}$ for the entailment relation with respect to the $\mathcal{C}^{\mathcal{B}}$ algebras and formulas from $\Sigma^{\mathcal{B}}$.

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, $q \not< p$, $\text{sort}(s) = S$, 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 $\mathcal{H} = (\mathcal{T}^H, \mathcal{T}^B)$, $\Rightarrow_{\mathsf{ABSTR}}$ terminates on N and preserves satisfiability. For any clause $C \in (N \Downarrow_{\mathsf{ABSTR}})$ 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 *foreground part* of the clause.

A *constrained closure* is denoted as $\Lambda \parallel C \cdot \sigma$ where σ is grounding for Λ and C. A constrained closure $\Lambda \parallel C \cdot \sigma$ denotes the ground constrained clause $\Lambda\sigma \parallel C\sigma$.

Running Example (BS(LRA) (1/2))

As a running example, I consider in detail the Bernays-Schoenfinkel clause fragment over linear arithmetic: BS(LRA).

The background theory is linear rational arithmetic over the many-sorted signature $\Sigma^{\mathsf{LRA}} = (\mathcal{S}^{\mathsf{LRA}}, \Omega^{\mathsf{LRA}}, \Pi^{\mathsf{LRA}})$ with $\mathcal{S}^{\mathsf{LRA}} = \{\mathsf{LRA}\}$, $\Omega^{\mathsf{LRA}} = \{0, 1, +, -\} \cup \mathbb{Q}$, $\Pi^{\mathsf{LRA}} = \{\leq, <, \neq, =, >, \geq\}$) where LRA is the linear arithmetic sort, the function symbols consist of $0, 1, +, -$ plus the rational numbers and predicate symbols $\leq, <, =, \neq, >, \geq$. The linear arithmetic theory $\mathcal{T}^{\mathsf{LRA}} = (\Sigma^{\mathsf{LRA}}, \{\mathcal{A}^{\mathsf{LRA}}\})$ consists of the linear arithmetic signature together with the standard model $\mathcal{A}^{\mathsf{LRA}}$ of linear arithmetic.

Running Example (BS(LRA) (2/2))

This theory is then extended by the free (foreground) first-order signature $\Sigma^{\mathsf{BS}} = (\{\mathsf{LRA}\}, \Omega^{\mathsf{BS}}, \Pi^{\mathsf{BS}})$ where Ω^{BS} is a set of constants of sort LRA different from Ω^{LRA} constants, and Π^{BS} is a set of first-order predicates over the sort LRA.

We are interested in hierarchic algebras $\mathcal{A}^{\mathsf{BS(LRA)}}$ over the signature $\Sigma^{\mathsf{BS(LRA)}} = (\{\mathsf{LRA}\}, \Omega^{\mathsf{BS}} \cup \Omega^{\mathsf{LRA}}, \Pi^{\mathsf{BS}} \cup \Pi^{\mathsf{LRA}})$ that are $\Sigma^{\mathsf{BS(LRA)}}$ algebras such that $\mathcal{A}^{\mathsf{BS(LRA)}}|_{\Sigma^{\mathsf{LRA}}} = \mathcal{A}^{\mathsf{LRA}}$.

- no foreground function symbols
- no constants in input clauses
- as usual all variables implicitly universally quantified
- all clauses are fully abstracted

In addition, we assume a well-founded, total, strict ordering $\prec$ on ground literals, called an $\mathcal{H}$-order, such that background literals are smaller than foreground literals. This ordering is then lifted to constrained clauses and sets thereof by its respective multiset extension. We overload $\prec$ for literals, constrained clauses, and sets of constrained clause if the meaning is clear from the context. We define $\preceq$ as the reflexive closure of $\prec$ and $N^{\preceq \Lambda \parallel C} := \{D \mid D \in N \text{ and } D \preceq \Lambda \parallel C\}$. For example, an instance of an LPO with according precedence can serve as $\prec$.

8.14.4 Definition (Clause Redundancy)

A ground constrained clause $\Lambda \parallel C$ is *redundant* with respect to a set N of ground constrained clauses and an order $\prec$ if $N^{\preceq \Lambda \parallel C} \models_{\mathcal{H}} \Lambda \parallel C$. A clause $\Lambda \parallel C$ is *redundant* with respect to a clause set N, an $\mathcal{H}$-order $\prec$, and a set of constants B if for all $\Lambda' \parallel C' \in \mathrm{grd}((\mathcal{S}^{\mathcal{F}}, B, \Pi^{\mathcal{B}} \cup \Pi^{\mathcal{F}}), \Lambda \parallel C)$ the clause $\Lambda' \parallel C'$ is redundant with respect to $\cup_{D \in N} \mathrm{grd}((\mathcal{S}^{\mathcal{F}}, B, \Pi^{\mathcal{B}} \cup \Pi^{\mathcal{F}}), D)$.

SCL(T)

8.14.5 Assumption (Considered Input Clause Sets)

For the rest of this section I consider only pure, abstracted clause sets N. I assume that the background theory $\mathcal{T}^{\mathcal{B}}$ is term-generated, compact, contains an equality $\simeq$, and that all constants of the background signature are domain constants. I further assume that the set $\Omega^{\mathcal{F}}$ contains infinitely many constants for each background sort.

In order for the SCL(T) calculus to be effective, decidability in $\mathcal{T}^{\mathcal{B}}$ is needed as well. For the calculus we implicitly use the following equivalence: A $\Sigma^{\mathcal{B}}$ sentence

$$\exists x_1, \ldots, x_n \phi$$

where ϕ is quantifier free is true, i.e., $\models_{\mathcal{B}} \exists x_1, \ldots, x_n \phi$ iff the ground formula

$$\phi\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$$

where the a_i are $\Omega^{\mathcal{F}}$ constants of the respective background sorts is $\mathcal{H}$ satisfiable. Together with decidability in $\mathcal{T}^{\mathcal{B}}$ this guarantees decidability of the satisfiability of ground constraints from constrained clauses.

If not stated otherwise, satisfiability means satisfiability with respect to $\mathcal{H}$. The function $\mathrm{adiff}(B)$ for some finite sequence of background sort constants denotes a constraint that implies different interpretations for the constants in B. In case the background theory enables a strict ordering $<$ as LRA does, then the ordering can be used for this purpose. For example, $\mathrm{adiff}([a, b, c])$ is then the constraint $a < b < c$. In case the background theory does not enable a strict ordering, then inequations can express disjointness of the constants. For example, $\mathrm{adiff}([a, b, c])$ is then constraint $a \neq b \wedge a \neq c \wedge b \neq c$. An ordering constraint has the advantage over an inequality constraint that it also breaks symmetries. Assuming all constants to be different will eventually enable a satisfiability test for foreground literals based on purely syntactic complementarity.

The inference rules of SCL(T) are represented by an abstract rewrite system. They operate on a problem state, a six-tuple $\Gamma = (M; N; U; B; k; D)$ where M is a sequence of annotated ground literals, the *trail*; N and U are the sets of *initial* and *learned* constrained clauses; B is a finite sequence of constants of background sorts for instantiation; k counts the number of decisions in M; and D is a constrained closure that is either $\top$, $\Lambda \parallel \bot \cdot \sigma$, or $\Lambda \parallel C \cdot \sigma$. Foreground literals in M are either annotated with a number, a level; i.e., they have the form L^k meaning that L is the k-th guessed decision literal, or they are annotated with a constrained closure that propagated the literal to become true, i.e., they have the form $(L\sigma)^{(\Lambda \parallel C \vee L)\cdot\sigma}$. An annotated literal is called a decision literal if it is of the form L^k and a propagation literal or a propagated literal if it of in the form $L \cdot \sigma^{(\Lambda \parallel C \vee L)\cdot\sigma}$.

A ground foreground literal L is of *level i* with respect to a problem state $(M; N; U; B; k; D)$ if L or $\mathrm{comp}(L)$ occurs in M and the first decision literal left from L ($\mathrm{comp}(L)$) in M, including L, is annotated with i. If there is no such decision literal then its level is zero. A ground constrained clause $\Lambda \parallel C$ is of *level i* with respect to a problem state $(M; N; U; B; k; D)$ if i is the maximal level of a foreground literal in C; the level of an empty clause $\Lambda \parallel \bot \cdot \sigma$ is 0.

A ground literal L is *undefined* in M if neither L nor $\mathrm{comp}(L)$ occur in M. The initial state for a first-order, pure, abstracted $\mathcal{H}$ clause set N is $(\epsilon; N; \emptyset; B; 0; \top)$, where B is a finite sequence of foreground constants of background sorts. These constants cannot occur in N, because N is pure. The final state $(\epsilon; N; U; B; 0; \Lambda \parallel \bot)$ denotes unsatisfiability of N. Given a trail M and its foreground literals $\mathrm{fgd}(M) = \{L_1, \ldots, L_n\}$ an $\mathcal{H}$ ordering $\prec$ *induced* by M is any $\mathcal{H}$ ordering where $L_i \prec L_j$ if L_i occurs left from L_j in M, or, L_i is defined in M and L_j is not.

Propagate $(M; N; U; B; k; \top) \Rightarrow_{\text{SCL(T)}}$
$(M, L\sigma^{(\Lambda \| C_0 \vee L)\delta \cdot \sigma}, \Lambda'\sigma; N; U; B; k; \top)$

provided $\Lambda \parallel C \in (N \cup U)$, σ is grounding for $\Lambda \parallel C$, $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, $C = C_0 \vee C_1 \vee L$, $C_1\sigma = L\sigma \vee \ldots \vee L\sigma$, $C_0\sigma$ does not contain $L\sigma$, δ is the mgu of the literals in C_1 and L, $\Lambda'\sigma$ are the background literals from $\Lambda\sigma$ that are not yet on the trail, $\text{fgd}(M) \models \neg(C_0\sigma)$, $\text{codom}(\sigma) \subseteq B$, and $L\sigma$ is undefined in M

The rule Propagate applies exhaustive factoring to the propagated literal with respect to the grounding substitution σ and annotates the factored clause to the propagation. By writing $M, L\sigma^{(\Lambda \| C_0 \vee L)\delta \cdot \sigma}, \Lambda'\sigma$ we denote that all background literals from $\Lambda'\sigma$ are added to the trail.

Decide $(M; N; U; B; k; \top) \Rightarrow_{\mathsf{SCL(T)}} (M, L\sigma^{k+1}, \Lambda\sigma; N; U; B; k+1; \top)$

provided $L\sigma$ is undefined in M, $|L\sigma| \in \mathsf{atoms}(\mathsf{grd}((\mathcal{S}, B, \Pi), N \cup U))$, $|K\sigma| \in \mathsf{atoms}(\mathsf{grd}((\mathcal{S}, B, \Pi), N \cup U))$ for all $K\sigma \in \Lambda\sigma$, σ is grounding for Λ, all background literals in $\Lambda\sigma$ are undefined in M, $\mathsf{adiff}(B) \wedge \mathsf{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, and $\mathsf{codom}(\sigma) \subseteq B$

Making sure that no duplicates of background literals occur on the trail by rules Propagate and Decide together with a fixed finite sequence B of constants and the restriction of Propagate and Decide to undefined literals guarantees that the number of potential trails of a run is finite. Requiring the constants from B to be different by the $\mathsf{adiff}(B)$ constraint enables a purely syntactic consistency check for foreground literals.

Conflict $(M; N; U; B; k; \top) \Rightarrow_{\text{SCL(T)}} (M; N; U; B; k; \Lambda \parallel D \cdot \sigma)$

provided $\Lambda \parallel D \in (N \cup U)$, σ is grounding for $\Lambda \parallel D$, $\text{adiff}(B) \wedge \text{bgd}(M) \wedge \Lambda\sigma$ is satisfiable, $\text{fgd}(M) \models \neg(D\sigma)$, and $\text{codom}(\sigma) \subseteq B$

Resolve $(M, L\rho^{\Lambda \parallel C \vee L \cdot \rho}; N; U; B; k; (\Lambda' \parallel D \vee L') \cdot \sigma) \Rightarrow_{\text{SCL(T)}} (M, L\rho^{\Lambda \parallel C \vee L \cdot \rho}; N; U; B; k; (\Lambda \wedge \Lambda' \parallel D \vee C)\eta \cdot \sigma\rho)$

provided $L\rho = \text{comp}(L'\sigma)$, and $\eta = \text{mgu}(L, \text{comp}(L'))$

Note that Resolve does not remove the literal $L\rho$ from the trail. This is needed if the clause $D\sigma$ contains further literals complementary of $L\rho$ that have not been factorized.

Factorize $(M; N; U; B; k; (\Lambda \parallel D \vee L \vee L') \cdot \sigma) \Rightarrow_{\text{SCL(T)}} (M; N; U; B; k; (\Lambda \parallel D \vee L)\eta \cdot \sigma)$

provided $L\sigma = L'\sigma$, and $\eta = \text{mgu}(L, L')$

Note that Factorize is not limited with respect to the trail. It may apply to any two literals that become identical by application of the grounding substitution σ.

Skip $(M, L; N; U; B; k; \Lambda' \parallel D \cdot \sigma) \Rightarrow_{\mathrm{SCL(T)}} (M; N; U; B; l; \Lambda' \parallel D \cdot \sigma)$

provided L is a foreground literal and $\mathrm{comp}(L)$ does not occur in $D\sigma$, or L is a background literal; if L is a foreground decision literal then $l = k - 1$, otherwise $l = k$

Note that Skip can also skip decision literals. This is needed because we won't eventually require exhaustive propagation. While exhaustive propagation in CDCL is limited to the number of propositional variables, in the context of our logic, for example $\mathrm{BS(LRA)}$, it is exponential in the arity of foreground predicate symbols and can lead to an unfair exploration of the space of possible inferences, harming completeness, see Example 8.14.9.

Backtrack $(M, K^{i+1}, M'; N; U; B; k; (\Lambda \parallel D \vee L) \cdot \sigma) \Rightarrow_{\mathsf{SCL(T)}} (M, L\sigma^{(\Lambda \parallel D \vee L) \cdot \sigma}, \Lambda'\sigma; N; U \cup \{\Lambda \parallel D \vee L\}; B; i; \top)$

provided $L\sigma$ is of level k, and $D\sigma$ is of level i, $\Lambda'\sigma$ are the background literals from $\Lambda\sigma$ that are not yet on the trail

The definition of Backtrack requires that if $L\sigma$ is the only literal of level k in $(D \vee L)\sigma$ then additional occurrences of $L\sigma$ in D have to be factorized first before Backtrack can be applied.

Grow $(M; N; U; B; k; \top) \Rightarrow_{\mathsf{SCL(T)}} (\epsilon; N; U; B \cup B'; 0; \top)$

provided B' is a non-empty sequence of foreground constants of background sorts distinct from the constants in B

In case the adiff constraint is implemented by a strict ordering predicate on the basis of the sequence B, it can be useful to inject the new constants B' into $B \cup B'$ such that the ordering of the constants from B is not changed. This can help caching background theory results for testing trail satisfiability.

8.14.9 Example (Exhaustive Propagation)

Consider a BS(LRA) clause set $N = \{x = 0 \parallel \mathrm{Nat}(x),\ y = x + 1 \parallel \neg\,\mathrm{Nat}(x) \vee \mathrm{Nat}(y)\} \cup N'$ where N' is unsatisfiable and nothing can be propagated from N'. Let us further assume that N' is satisfiable with respect to any instantiation of variables with natural numbers. If propagation is not restricted, then the first two clauses will consume all constants in B. For example, if $B = [a, b, c]$ then the trail $[\mathrm{Nat}(a), a = 0, \mathrm{Nat}(b), b = a + 1, \mathrm{Nat}(c), c = b + 1]$ will be derived. Now all constants are fixed to natural numbers. So there cannot be a refutation of N' anymore. An application of Grow will not solve the issue, because again the first two rules will fix all constants to natural numbers via exhaustive propagation.

8.14.10 Definition (Well-formed States)

A state $(M; N; U; B; k; D)$ is *well-formed* if the following conditions hold:

1. all constants appearing in $(M; N; U; B; k; D)$ are from B or occur in N.
2. $M \wedge \mathrm{adiff}(B)$ is satisfiable
3. $N \models_{\mathcal{H}} U$,
4. Propagating clauses remain propagating and conflict clauses remain false:
   - 1..1 if $D = \Lambda \parallel C \cdot \sigma$ then $C\sigma$ is false in $\mathrm{fgd}(M)$ and $\mathrm{bgd}(M) \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable,
   - 2..2 if $M = M_1, L\sigma^{(\Lambda \parallel C \vee L)\cdot\sigma}, M_2$ then $C\sigma$ is false in $\mathrm{fgd}(M_1)$, $L\sigma$ is undefined in M_1, and $\mathrm{bgd}(M_1) \wedge \mathrm{adiff}(B) \wedge \Lambda\sigma$ is satisfiable.
5. All clauses in $N \cup U$ are pure. In particular, they don't contain any constants from B.

8.14.11 Lemma (Rules preserve Well-Formed States)

The rules of SCL(T) preserve well-formed states.

8.14.12 Definition (Stuck State)

A state $(M; N; U; B; k; D)$ is called *stuck* if $D \neq \Lambda \parallel \bot \cdot \sigma$ and none of the rules Propagate, Decide, Conflict, Resolve, Factorize, Skip, or Backtrack is applicable.

8.14.13 Proposition (Form of Stuck States)

If a run (without rule Grow) ends in a stuck state $(M; N; U; B; k; D)$ where Conflict was applied eagerly, then $D = \top$ and all ground foreground literals that can be build from the foreground literals in N by instantiation with constants from B are defined in M.

Lemma (Stuck States Produce Ground Models)

If a state $(M; N; U; B; k; \top)$ is stuck then $M \wedge \mathsf{adiff}(B) \models \mathsf{grd}((\mathcal{S}, B, \Pi), N \cup U)$.

8.14.15 Example (SCL(T) Model Extraction)

In some cases it is possible to extract an overall model from the ground trail of a stuck state of an SCL(T) derivation. Consider $B = [a, b, c]$ and a satisfiable BS(LRA) constrained clause set $N = \{x \geq 1 \parallel P(x), x < 0 \parallel P(x), 0 \leq x \wedge x < 1 \parallel \neg P(x), 2x \geq 1 \parallel P(x) \vee Q(x)\}$. Starting from state $(\epsilon; N; \emptyset; B; 0; \top)$ and applying Propagate fairly a regular run can derive the following trail

$$M = P(a)^{x \geq 1 \parallel P(x) \cdot \{x \mapsto a\}}, a \geq 1, P(b)^{x < 0 \parallel P(x) \cdot \{x \mapsto b\}}, b < 0, \neg P(c)^{0 \leq x \wedge x < 1 \parallel \neg P(x) \cdot \{x \mapsto c\}}, 0 \leq c, c < 1, Q(c)^{2x \geq 1 \parallel P \vee Q(x) \cdot \{x \mapsto c\}},$$

The state $(M; N; \emptyset; B; 0; \top)$ is stuck and $M \models_{\mathcal{H}} \mathrm{grd}((\mathcal{S}, B, \Pi), N)$. Moreover from M we can generate an interpretation $\mathcal{A}^{\mathrm{BS(LRA)}}$ of N by generalizing the foreground constants used for instantiation and interpreting the predicates P and Q as formulas over $\Sigma^{\mathcal{B}}$, $P^{\mathcal{A}} = \{q \in \mathbb{Q} \mid q < 0 \vee q \geq 1\}$ and $Q^{\mathcal{A}} = \{q \in \mathbb{Q} \mid 2q \geq 1 \wedge q < 1\}$.

8.14.16 Lemma (Soundness)

If a derivation reaches the state $(M; N; U; B; k; \Lambda \parallel \bot \cdot \sigma)$, then N is unsatisfiable.

8.14.17 Definition (Reasonable Run)

A sequence of SCL(T) rule applications is called a *reasonable run* if the rule Decide is only applied if there exists no application of the rule Propagate that would generate a conflict.

8.14.18 Definition (Regular Run)

A sequence of SCL(T) rule applications is called a *regular run* if it is a reasonable run the rule Conflict has precedence over all other rules, and Resolve resolves away at least the rightmost foreground literal from the trail.

8.14.23 Lemma (Non-Redundant Clause Learning)

Let N be a set of constrained clauses. Then clauses learned in an SCL(T) regular run from starting state $(\epsilon; N; \emptyset; B; 0; \top)$ are not redundant.

8.14.24 Lemma (Termination of SCL(T))

Let N be a set of constrained clauses and B be a finite set of background constants. Then any regular run with start state $(\epsilon; N; \emptyset; B; 0; \top)$ that uses Grow only finitely often terminates.

8.14.25 Theorem (Hierarchic Herbrand Theorem)

Let N be a finite set of clauses. N is unsatisfiable iff there exists a finite set $N' = \{\Lambda_1 \parallel C_1, \ldots, \Lambda_n \parallel C_n\}$ of variable renamed copies of clauses from N and a finite set B of fresh constants and a substitution σ, grounding for N' where $\mathrm{codom}(\sigma) = B$ such that $\bigwedge_i \Lambda_i\sigma$ is $\mathcal{T}^{\mathcal{B}}$ satisfiable and $\bigwedge_i C_i\sigma$ is first-order unsatisfiable over $\Sigma^{\mathcal{F}}$.

8.14.26 Theorem (Refutational Completeness of SCL(T))

Let N be an unsatisfiable clause set. Then any regular SCL(T) run will derive the empty clause provided (i) Rule Grow and Decide are operated in a fair way, such that all possible trail prefixes of all considered sets B during the run are eventually explored, and (ii) Restart is only applied to stuck states.