Decision Procedures in Verification

Decision Procedures (3)

7.1.2013

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Until now:

Logical Theories: generalities

Theory of Uninterpreted Function Symbols

DAG representation of terms/Congruence closure of DAGs

Decision procedures for numeric domains

brief mention of undecidability results

brief mention of decidability of the theory of real closed fields

Linear arithmetic: definition

- simple fragment of linear arithmetic: Difference logic

Positive difference logic: Reminder

Syntax

The syntax of formulae in positive difference logic is defined as follows:

- Atomic formulae (also called difference constraints) are of the form:

  $$x - y \leq c$$

  where x, y are variables and c is a numerical constant.

- The set of formulae is:

$$\begin{array}{lcll} F, G, H & ::= & \mathsf{A} & \text{(atomic formula)} \\ & | & (F \wedge G) & \text{(conjunction)} \end{array}$$

Semantics:

Versions of difference logic exist, where the standard interpretation is $\mathbb{Q}$ or resp. $\mathbb{Z}$.

Positive difference logic: Reminder

A decision procedure for positive difference logic ($\leq$ only)

Let S be a set (i.e. conjunction) of atoms in (positive) difference logic. $G(S) = (V, E, w)$, the inequality graph of S, is a weighted graph with:

- $V = X(S)$, the set of variables occurring in S
- $e = (x, y) \in E$ with $w(e) = c$ iff $x - y \leq c \in S$

Theorem 3.4.1.
Let S be a conjunction of difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Searching for negative cycles in a graph can be done with the Bellman-Ford algorithm for finding the single-source shortest paths in a directed weighted graph in time $O(|V| \cdot |E|)$. (Side-effect of the algorithm exploited - if there exists a negative cycle in the graph then the algorithm finds it and aborts.)

Positive difference logic: Reminder

Theorem 3.4.1.
Let S be a conjunction of difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Proof: ($\Rightarrow$) Assume S satisfiable. Let $\beta : X \to \mathbb{Z}$ satisfying assignment.
Let $v_1 \overset{c_{12}}{\to} v_2 \overset{c_{23}}{\to} \cdots \overset{c_{n-1,n}}{\to} v_n \overset{c_{n1}}{\to} v_1$ be a cycle in $G(S)$.

$$
\begin{array}{rrcl}
\text{Then:} & \beta(v_1) - \beta(v_2) & \leq & c_{12} \\
 & \beta(v_2) - \beta(v_3) & \leq & c_{23} \\
 & \cdots & & \\
g & \beta(v_n) - \beta(v_1) & \leq & c_{n1} \\
\hline
0 = & \beta(v_1) - \beta(v_1) & \leq & \sum_{i=1}^{n-1} c_{i,i+1} + c_{n1}
\end{array}
$$

Thus, for satisfiability it is necessary that all cycles are positive.

Positive difference logic

Theorem 3.4.1.
Let S be a conjunction of difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Proof: ($\Leftarrow$) Assume that there is no negative cycle.

Add a new vertex s and an 0-weighted edge from every vertex in V to s. (This does not introduce negative cycles.)

Let δ_{uv} denote the minimal weight of the paths from u to v.

- $\delta_{uv} = \infty$ if there is no path from u to v.
- well-defined since there are no negative cycles

Define $\beta : V \to \mathbb{Z}$ by $\beta(v) = \delta_{vs}$. Claim: β satisfying assignment for S.

Let $x - y \leq c \in S$. Consider the shortest paths from x to s and from y to s. By the triangle inequality, $\delta_{xs} \leq c + \delta_{ys}$, i.e. $\beta(x) - \beta(y) \leq c$.

Difference logic

Syntax

- Atomic formulae (difference constraints): $x - y \leq c$

  where x, y are variables and c is a numerical constant.

- Formulae:

$$
\begin{array}{llll}
F, G, H & ::= & \mathrm{A} & \text{(atomic formula)} \\
 & \mid & \neg A & \\
 & \mid & (F \wedge G) & \text{(conjunction)}
\end{array}
$$

Note: $\neg(x - y \leq c)$ is equivalent to $y - x < c$.

Difference logic

Syntax

- Atomic formulae (difference constraints): $x - y \leq c$

  where x, y are variables and c is a numerical constant.

- Formulae:

$$
\begin{array}{llll}
F, G, H & ::= & A & \text{(atomic formula)} \\
 & | & \neg A & \\
 & | & (F \wedge G) & \text{(conjunction)}
\end{array}
$$

Note: $\neg(x - y \leq c)$ is equivalent to $y - x < c$.

Satisfiability over $\mathbb{Z}$

$y - x < c$ iff $y - x \leq c - 1$

Natural reduction to positive difference logic.

Difference logic

Syntax

- Atomic formulae (difference constraints): $x - y \leq c$

  where x, y are variables and c is a numerical constant.

- Formulae:

$$
\begin{array}{llll}
F, G, H & ::= & A & \text{(atomic formula)} \\
 & | & \neg A & \\
 & | & (F \wedge G) & \text{(conjunction)}
\end{array}
$$

Note: $\neg(x - y \leq c)$ is equivalent to $y - x < c$.

Theorem 3.4.2.
Let S be a conjunction of strict and non-strict difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Difference logic

Theorem 3.4.2.
Let S be a conjunction of strict and non-strict difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Proof:
Need to extend the graph construction and the unsatisfiability condition:

$x_1 - x_2 \prec_1 c_1, \ldots, x_n - x_1 \prec_n c_n$ unsatisfiable iff

- $\sum_{i=1}^{n} c_i < 0$, or
- $\sum_{i=1}^{n} c_i = 0$ and one $\prec_i$ is strict.

Consider pairs $(\prec, c)$ instead of numbers c

- $(\prec, c) <_B (\prec', c')$ iff $c < c'$ or ($c = c'$, $\prec_1 = <$ and $\prec_2 = \leq$)
- $(\prec, c) + (\prec', c') = (\prec'', c + c')$ where $\prec'' = <$ iff $\prec$ or $\prec'$ is $<$.

Linear arithmetic over $\mathbb{N}$ or $\mathbb{Z}$

1. $\mathrm{Th}(\mathbb{Z}_+)$ $\quad$ $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +, <)$ the standard interpretation of integers.
2. **Presburger arithmetic.**

Axiomatization:

$\forall x\, \neg(x + 1 \approx 0)$	(zero)
$\forall x \forall y\, (x + 1 \approx y + 1 \rightarrow x \approx y$	(successor)
$F[0] \wedge (\forall x\, (F[x] \rightarrow F[x + 1]) \rightarrow \forall x F[x])$	(induction)
$\forall x\, (x + 0 \approx x)$	(plus zero)
$\forall x, y\, (x + (y + 1) \approx (x + y) + 1)$	(plus successor)

Linear arithmetic over $\mathbb{N}$ or $\mathbb{Z}$

Presburger arithmetic decidable in 3EXPTIME [Presburger'29]

- automata theoretic method

Linear arithmetic over $\mathbb{Z}$:

check satisfiability of conjunctions of equalities over $\mathbb{Z}$: NP-hard

- Integer linear programming

  use branch-and-bound/cutting planes

- The Omega test – use variable elimination

Linear arithmetic over $\mathbb{R}$ or $\mathbb{Q}$

- $\mathsf{Th}(\mathbb{R})$
  $\mathbb{R} = (\mathbb{R}, \{0, 1, +\}, \{<\})$ the standard interpretation of real numbers;
- $\mathsf{Th}(\mathbb{Q})$
  $\mathbb{Q} = (\mathbb{Q}, \{0, 1, +\}, \{<\})$ the standard interpretation of rational numbers.

Linear arithmetic over $\mathbb{R}$ or $\mathbb{Q}$

Theorem.

(1) The satisfiability of any conjunction of (strict and non-strict) linear inequalities can be checked in PTIME [Khakian'79].

(2) The complexity of checking the satisfiability of sets of clauses in linear arithmetic is in NP [Sonntag'85].

Literature

[Khakian'79] L. Khachian. "A polynomial time algorithm for linear programming." *Soviet Math. Dokl.* 20:191-194, 1979.

[Sonntag'85] E.D. Sontag. "Real addition and the polynomial hierarchy". *Inf. Proc. Letters* 20(3):115-120, 1985.

Linear arithmetic over $\mathbb{R}$ or $\mathbb{Q}$

Methods The algorithms currently used are not PTIME.

- The simplex method
- The Fourier-Motzkin method – use variable elimination

Linear arithmetic: Comparison

Problem:

check satisfiability of conjunctions of equalities over a numerical domain D

Complexity: $D = \mathbb{R}$: PTIME; $D = \mathbb{Z}$: NP-hard

Methods

- The simplex method ($D = \mathbb{R}$)
- Integer linear programming ($D = \mathbb{Z}$)

  use branch-and-bound/cutting planes

- The Fourier-Motzkin method ($D = \mathbb{R}$)

  use variable elimination

- The Omega test ($D = \mathbb{Z}$)

  use variable elimination

Linear arithmetic: Comparison

Problem:

check satisfiability of conjunctions of equalities over a numerical domain D

Complexity: $D = \mathbb{R}$: PTIME; $D = \mathbb{Z}$: NP-hard

Methods

- The simplex method ($D = \mathbb{R}$)
- Integer linear programming ($D = \mathbb{Z}$)

  use branch-and-bound/cutting planes

- The Fourier-Motzkin method ($D = \mathbb{R}$) Today

  use variable elimination

- The Omega test ($D = \mathbb{Z}$)

  use variable elimination

Linear arithmetic over $\mathbb{R}$ or $\mathbb{Q}$

Axiomatization:

The equational part of linear rational arithmetic is described by the theory of divisible torsion-free abelian groups:

$$\forall x, y, z(x + (y + z) \approx (x + (y + z))) \qquad \text{(associativity)}$$

$$\forall x, y(x + y \approx y + x) \qquad \text{(commutativity)}$$

$$\forall x(x + 0 \approx x) \qquad \text{(identity)}$$

$$\forall x \exists y(x + y \approx 0) \qquad \text{(inverse)}$$

$$\text{For all } n \geq 1\text{: } \forall x(\underbrace{x + \cdots + x}_{n \text{ times}} \approx 0 \rightarrow x \approx 0) \qquad \text{(torsion-freeness)}$$

$$\text{For all } n \geq 1 : \forall x \exists y(\underbrace{y + \cdots + y}_{n \text{ times}} \approx x) \qquad \text{(divisibility)}$$

$$\neg 1 \approx 0 \qquad \text{(non-triviality)}$$

Note: Quantification over natural numbers is not part of our language. We really need infinitely many axioms for torsion-freeness and divisibility.

Linear arithmetic over $\mathbb{R}$ or $\mathbb{Q}$

By adding the axioms of a compatible strict total ordering, we define ordered divisible abelian groups (ODAG):

$$\begin{aligned}
\forall x\ (\neg x < x) &\qquad \text{(irreflexivity)}\\
\forall x, y, z\ (x < y \wedge y < z \rightarrow x < z) &\qquad \text{(transitivity)}\\
\forall x, y\ (x < y \vee y < x \vee x \approx y) &\qquad \text{(totality)}\\
\forall x, y, z\ (x < y \rightarrow x + z < y + z) &\qquad \text{(compatibility)}\\
0 < 1 &\qquad \text{(non-triviality)}
\end{aligned}$$

Note: The second non-triviality axiom renders the first one superfluous.

Moreover, as soon as we add the axioms of compatible strict total orderings, torsion-freeness can be omitted.

Every ordered divisible abelian group is obviously torsion-free. In fact the converse holds: Every torsion-free abelian group can be ordered [F.-W. Levi, 1913].

Examples: $\mathbb{Q}, \mathbb{R}, \mathbb{Q}^n, \mathbb{R}^n, \ldots$

Linear arithmetic over $\mathbb{R}$ or $\mathbb{Q}$

The signature can be extended by further symbols:

- $\leq /2, > /2, \geq /2, \not\approx /2$: defined using $<$ and $\approx$
- $-/1$: Skolem function for inverse axiom
- $-/2$: defined using $+/2$ and $-/1$
- $\mathrm{div_n}/1$: Skolem functions for divisibility axiom for all $n \geq 1$.
- $\mathrm{mult_n}/1$: defined by $\forall x(\mathrm{mult_n}(x) \approx \underbrace{x + \cdots + x}_{n \text{ times}}$ for all $n \geq 1$.
- $\mathrm{mult_q}/1$: defined using $\mathrm{mult_n}, \mathrm{div_n}, -$ for all $q \in \mathbb{Q}$.
  (We usually write $q \cdot t$ or qt instead of $\mathrm{mult_q}(t)$.)
- $q/0$ (for $q \in \mathbb{Q}$): defined by $q \approx q \cdot 1$.

Note: Every formula using the additional symbols is ODAG-equivalent to a formula over the base signature.

When $\cdot$ is considered as a binary operator, (ordered) divisible torsion-free abelian groups correspond to (ordered) rational vector spaces.

Fourier-Motzkin Quantifier Elimination

Linear rational arithmetic permits quantifier elimination:
every formula $\exists x F$ or $\forall x F$ in linear rational arithmetic can be converted into an equivalent formula without the variable x.

The method was discovered in 1826 by J. Fourier and re-discovered by T. Motzkin in 1936.

Observation: Every literal over the variables $x, y_1, \ldots, y_n$ can be converted into an ODAG-equivalent atom $x \sim t[\overline{y}]$ or $0 \sim t[\overline{y}]$, where $\sim \in \{<, >, \leq, \geq, \approx, \not\approx\}$ and $t[\overline{y}]$ has the form $\sum_i q_i \cdot y_i + q_0$.

In other words, we can either eliminate x completely or isolate it on one side of the atom.

Moreover, we can convert every $\not\approx$ atom into an ODAG-equivalent disjunction of two $<$ atoms.

Fourier-Motzkin Quantifier Elimination

We first consider existentially quantified conjunctions of atoms.

(1) **If the conjunction contains an equation** $x \approx t[\overline{y}]$,
we can eliminate the quantifier $\exists x$ by substitution:

$$\exists x(x \approx t[\overline{y}] \wedge F)$$

is equivalent to

$$F\sigma, \text{ where } \sigma = [t[\overline{y}]/x]$$

Fourier-Motzkin Quantifier Elimination

We first consider existentially quantified conjunctions of atoms.

(2) **If x occurs only in inequations, then:**

$$\begin{aligned}\exists x \quad & (\textstyle\bigwedge_i x < s_i(\overline{y}) \wedge \bigwedge_j x \leq t_j(\overline{y}) \wedge \\ & \textstyle\bigwedge_k x > u_k(\overline{y}) \wedge \bigwedge_l x \geq v_l(\overline{y}) \wedge \\ & F(\overline{y}))\end{aligned}$$

is equivalent to:

$$\begin{aligned}& \textstyle\bigwedge_i \bigwedge_k s_i(\overline{y}) > u_k(\overline{y}) \wedge \bigwedge_j \bigwedge_k t_j(\overline{y}) > u_k(\overline{y}) \wedge \\ & \textstyle\bigwedge_i \bigwedge_l s_i(\overline{y}) > v_l(\overline{y}) \wedge \bigwedge_j \bigwedge_l t_j(\overline{y}) \geq v_l(\overline{y}) \wedge \\ & F(\overline{y})\end{aligned}$$

Proof: "$\Rightarrow$" follows by transitivity;

"$\Leftarrow$" Take $\frac{1}{2}(\min\{s_i, t_j\} + \max\{u_k, v_l\})$ as a witness.

Fourier-Motzkin Quantifier Elimination

Extension to arbitrary formulas:

- Transform into prenex formula;
- If innermost quantifier is $\exists$:
  transform matrix into DNF and move $\exists$ into disjunction;
- If innermost quantifier is $\forall$: replace $\forall x F$ by $\neg\exists x \neg F$, then eliminate $\exists$.

Consequences:

(1) Every closed formula over the signature of ODAGs is ODAG-equivalent to either $\top$ or $\bot$.

(2) ODAGs are a complete theory, i.e., every closed formula over the signature of ODAGs is either valid or unsatisfiable w.r.t. ODAGs.

(3) Every closed formula over the signature of ODAGs holds either in all ODAGs or in no ODAG.

ODAGs are indistinguishable by first-order formulas over the signature of ODAGs. (These properties do not hold for extended signatures!)

Fourier-Motzkin: Complexity

- **One FM-step for $\exists$:**

  formula size grows quadratically, therefore $O(n^2)$ runtime.

- ***m* quantifiers $\exists \ldots \exists$:**

  naive implementation needs $O(n^{2^m})$ runtime;

  It is unknown whether optimized implementation with simply exponential runtime is possible.

- ***m* quantifiers $\exists\forall\exists\forall \ldots \exists\forall$:**

  CNF/DNF conversion (exponential!) required after each step; therefore non-elementary runtime.

Fourier-Motzkin: Complexity

- **One FM-step for $\exists$:**
  formula size grows quadratically, therefore $O(n^2)$ runtime.
- **m quantifiers $\exists \ldots \exists$:**
  naive implementation needs $O(n^{2^m})$ runtime;

  It is unknown whether optimized implementation with simply exponential runtime is possible.
- **m quantifiers $\exists\forall\exists\forall \ldots \exists\forall$:**
  CNF/DNF conversion (exponential!) required after each step; therefore non-elementary runtime.

Improvement: Loos-Weispfenning Quantifier Elimination

(will not be presented in this lecture)

3.5. Combinations of theories

Motivation

Program Verification

Example: Does BubbleSort return a sorted array?

```
int [] BubbleSort(int[] a) {
    int i, j, t;
    for (i := |a| − 1; i > 0; i := i − 1) {
      for (j := 0; j < i; j := j + 1) {
        if (a[j] > a[j + 1]){t := a[j];
                              a[j] := a[j + 1];
                              a[j + 1] := t};
}} return a}
```

Motivation

Example: Does BubbleSort return a sorted array?

```
int [] BubbleSort(int[] a) {
    int i, j, t;
    for (i := |a| − 1; i > 0; i := i − 1) {
        for (j := 0; j < i; j := j + 1) {
            if (a[j] > a[j + 1]){t := a[j];
                                 a[j] := a[j + 1];
                                 a[j + 1] := t};
}} return a}
```

$-1 \leq i < |a| \wedge$
$\text{partitioned}(a, 0, i, i+1, |a| - 1) \wedge$
$\text{sorted}(a, i, |a| - 1)$

$-1 \leq i < |a| \wedge 0 \leq j \leq i \wedge$
$\text{partitioned}(a, 0, i, i+1, |a| - 1) \wedge$
$\text{sorted}(a, i, |a| - 1)$
$\text{partinioned}(a, 0, j-1, j, j)$

Generate verification conditions and prove that they are valid

Predicates:

- $\text{sorted}(a, l, u)$: $\forall i, j (l \leq i \leq j \leq u \rightarrow a[i] \leq a[j])$
- $\text{partitioned}(a, l_1, u_1, l_2, u_2)$: $\forall i, j (l_1 \leq i \leq u_1 \leq l_2 \leq j \leq u_2 \rightarrow a[i] \leq a[j])$

Motivation

$-1 \leq i < |a| \wedge$ $\boxed{C_1(a)}$

$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$

$\text{sorted}(a, i, |a|-1)$

$-1 \leq i < |a| \wedge 0 \leq j \leq i \wedge$ $\boxed{C_2(a)}$

$\text{partitioned}(a, 0, i, i+1, |a|-1) \wedge$

$\text{sorted}(a, i, |a|-1)$

$\text{partinioned}(a, 0, j-1, j, j)$

Example: Does BubbleSort return a sorted array?

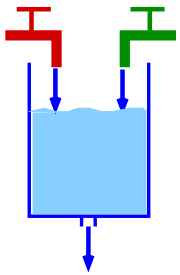
```
int [] BubbleSort(int[] a) {
    int i, j, t;
    for (i := |a| − 1; i > 0; i := i − 1) {
        for (j := 0; j < i; j := j + 1) {
            if (a[j] > a[j + 1]){t := a[j];
                                 a[j] := a[j + 1];
                                 a[j + 1] := t};
}} return a}
```

Generate verification conditions and prove that they are valid

$C_2(a) \wedge \text{Update}(a, a') \rightarrow C_2(a')$

Motivation

Verification of real time/hybrid systems

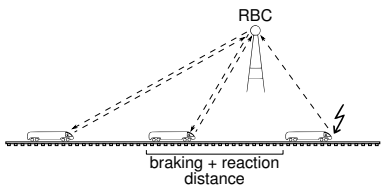


Check:

- No overflow
- Substances in the right proportion
- If substances in wrong proportion, tank can be drained in $\leq$ 200s.

Determine values for parameters such that this is the case

Train controllers



- **Task:** check collision freeness

Motivation

Mathematics

Example: Lipschitz functions

$$\boxed{\mathbb{R} \cup (\mathsf{L}^{\mathsf{f}}_{\mathsf{c},\lambda_1}) \cup (\mathsf{L}^{\mathsf{g}}_{\mathsf{c},\lambda_2}) \;\models\; (L^{f+g}_{c,(\lambda_1+\lambda_2)})}$$

$(\mathsf{L}^{\mathsf{f}}_{\mathsf{c},\lambda_1})$ $\forall x\ |f(x) - f(c)| \leq \lambda_1 \cdot |x - c|$

$(\mathsf{L}^{\mathsf{g}}_{\mathsf{c},\lambda_2})$ $\forall x\ |g(x) - g(c)| \leq \lambda_2 \cdot |x - c|$

$(\mathsf{L}^{\mathsf{f+g}}_{\mathsf{c},(\lambda_1+\lambda_2)})$ $\forall x\ |f(x)+g(x)-f(c)-g(c)| \leq (\lambda_1+\lambda_2) \cdot |x-c|$

Similar:
- free functions; (piecewise) monotone functions
- functions defined according to a partition of their domain of definition, ...

Combinations of theories

The combined validity problem

For $i = 1, 2$
- let $\mathcal{T}_i$ be a first-order theory in signature Σ_i
- let $\mathcal{L}_i$ be a class of (closed) Σ_i-formulae

Let $\mathcal{T}_1 \bigoplus \mathcal{T}_2$ be a combination of $\mathcal{T}_1$ and $\mathcal{T}_2$
Let $\mathcal{L}_1 \bigoplus \mathcal{L}_2$ be a combination of $\mathcal{L}_1$ and $\mathcal{L}_2$

Problem: Given ϕ in $\mathcal{L}_1 \bigoplus \mathcal{L}_2$, is it the case that $\mathcal{T}_1 \bigoplus \mathcal{T}_2 \models \phi$?

Problems

The combined decidability problem I

For $i = 1, 2$
- let $\mathcal{T}_i$ be a first-order theory in signature Σ_i
- let $\mathcal{L}_i$ be a class of (closed) Σ_i-formulae
- assume the $\mathcal{T}_i$-validity problem for $\mathcal{L}_i$ is decidable

Let $\mathcal{T}_1 \bigoplus \mathcal{T}_2$ be a combination of $\mathcal{T}_1$ and $\mathcal{T}_2$
Let $\mathcal{L}_1 \bigoplus \mathcal{L}_2$ be a combination of $\mathcal{L}_1$ and $\mathcal{L}_2$

Question: Is the $\mathcal{T}_1 \bigoplus \mathcal{T}_2$-validity problem for $\mathcal{L}_1 \bigoplus \mathcal{L}_2$ decidable?

Problems

The combined decidability problem II

For $i = 1, 2$
- let $\mathcal{T}_i$ be a first-order theory in signature Σ_i
- let $\mathcal{L}_i$ be a class of (closed) Σ_i-formulae
- P_i decision procedure for $\mathcal{T}_i$-validity for $\mathcal{L}_i$

Let $\mathcal{T}_1 \bigoplus \mathcal{T}_2$ be a combination of $\mathcal{T}_1$ and $\mathcal{T}_2$
Let $\mathcal{L}_1 \bigoplus \mathcal{L}_2$ be a combination of $\mathcal{L}_1$ and $\mathcal{L}_2$

Question: Can we combine P_1 and P_2 modularly into a decision procedure for the $\mathcal{T}_1 \bigoplus \mathcal{T}_2$-validity problem for $\mathcal{L}_1 \bigoplus \mathcal{L}_2$?

Main issue: How are $\mathcal{T}_1 \bigoplus \mathcal{T}_2$ and $\mathcal{L}_1 \bigoplus \mathcal{L}_2$ defined?

Combinations of theories and models

Forgetting symbols

Let $\Sigma = (\Omega, \Pi)$ and $\Sigma' = (\Omega', \Pi')$ s.t. $\Sigma \subseteq \Sigma'$, i.e., $\Omega \subseteq \Omega'$ and $\Pi \subseteq \Pi'$

For $\mathcal{A} \in \Sigma'$-alg, we denote by $\mathcal{A}_{|\Sigma}$ the Σ-structure for which:

$$U_{\mathcal{A}_{|\Sigma}} = U_{\mathcal{A}}, \quad f_{\mathcal{A}_{|\Sigma}} = f_{\mathcal{A}} \quad \text{for } f \in \Omega;$$
$$P_{\mathcal{A}_{|\Sigma}} = P_{\mathcal{A}} \quad \text{for } P \in \Pi$$

(ignore functions and predicates associated with symbols in $\Sigma' \backslash \Sigma$)

$\mathcal{A}_{|\Sigma}$ is called the restriction (or the reduct) of $\mathcal{A}$ to Σ.

Example: $\Sigma' = (\{+/2, */2, 1/0\}, \{\leq /2, \text{even}/1, \text{odd}/1\})$

$\Sigma = (\{+/2, 1/0\}, \{\leq /2\}) \subseteq \Sigma'$

$\mathcal{N} = (\mathbb{N}, +, *, 1, \leq, \text{even}, \text{odd})$ $\qquad$ $\mathcal{N}_{|\Sigma} = (\mathbb{N}, +, 1, \leq)$