The Model Evolution Calculus with Built-in Theories

Peter Baumgartner
MPI Informatik, Saarbrücken

www.mpi-sb.mpg.de/~baumgart/
Problem

- The Model Evolution Calculus is a sound and refutationally complete calculus for first-order clause logic
- **Can we extend it with built-in theory handling?**
  That is, „plug in“ an (efficient) reasoner for a special domain
- Examples for interesting theories
  - Equality
  - Real arithmetic
  - Theories axiomatized by logic programs
- **Can existing theory reasoners be plugged in (to Darwin)?**
  - Equality: Waldmeister
  - Real arithmetic: quantifier elimination
  - Logic programs: logic program interpreter
Model Evolution – Idea (1)

**DPLL**: Davis-Putnam-Logemann-Loveland Procedure (1960-63)
Basis of some of the SAT solvers (Chaff, …)

**Input**: Propositional clause set
**Output**: Model or „unsatisfiable”

**Algorithm components:**
- Simplification
- Split
- Backtracking

\[
\{A, B\} \models \{\neg A \lor \neg B \lor C \lor D, \ldots\}
\]

No, split on \(C\)

\[
\{A, B, C\} \models \{\neg A \lor \neg B \lor C \lor D, \ldots\}
\]
Model Evolution – Idea (2)

≈First Order DPLL [Joint Work with Cesare Tinelli]

Input: First-order clause set
Output: Model or „unsatisfiable” if termination

Procedure components:
- Simplification
- Split
- Backtracking

\{	ext{flight(sb,y), } \neg\text{flight(sb,d)}} \models \{\text{flight(x,y) } \lor \text{train(x,y), …}\}

No, split on train(sb,d)

\{\text{flight(sb,y), } \neg\text{flight(sb,d), train(sb,d)}} \models \{\text{flight(x,y) } \lor \text{train(x,y), …}\}
Calculus

- Sequent Style Calculus

\[ \Lambda \vdash \Phi \]

**Current Clause Set:**
Initially: input clauses

**Context:** A set of literals
(same as branch on previous slide)
Initially: \( \{ \neg v \} \)

- Simplified Calculus (for the purpose of talk)
  - No simplification inference rules to modify \( \Phi \)
  - No simplification inference rules to modify \( \Lambda \)
  - No „universal“ variables, only „parametric“ ones
Derivation Rules (1)

Split \( \frac{\Lambda \vdash \Phi, C \lor L}{\Lambda, L\sigma \vdash \Phi, C \lor L} \) 
\( \frac{\Lambda, \overline{L}\sigma \vdash \Phi, C \lor L}{\Lambda, \overline{\overline{L}}\sigma \vdash \Phi, C \lor L} \)

if
(1) \( \sigma \) is a context unifier of \( C \lor L \) against \( \Lambda \)
(2) neither \( L\sigma \) nor \( \overline{L}\sigma \) is contradictory with \( \Lambda \)

- \( \sigma \) is a context unifier: \( \sigma \) is a most general simultaneous unifier of the clause literals and context literals with opposite sign (pairwise)
- \( L\sigma \) is contradictory with \( \Lambda \): \( \Lambda \) contains a variant of \( \overline{L}\sigma \)

Context: \( P(u,u) \quad Q(v,b) \)
Clause: \( \overline{P(x,y)} \lor \overline{Q(a,z)} \)

\( \sigma = \{ x \mapsto u, y \mapsto u, v \mapsto a, z \mapsto b \} \)

Clause \( \sigma \): \( \overline{P(x,x)} \lor \overline{Q(a,b)} \)
\( \overline{Q(a,b)} \) is admissible for Split

contradictory \quad not contradictionary
Derivation Rules (2)

\[
\begin{align*}
\text{Close} & \quad \frac{\Lambda \vdash \Phi, C}{\Lambda \vdash \bot} \\
\text{if (1) } & \quad \Phi \neq \emptyset \text{ or } C \neq \bot \\
\text{(2) there is a context unifier } & \quad \sigma \text{ of } C \text{ against } \Lambda \\
\text{such that each literal of } C\sigma \text{ is contradictory with } \Lambda
\end{align*}
\]

- \( \sigma \) is a **context unifier**: \( \sigma \) is a most general simultaneous unifier of the clause literals and context literals with opposite sign (pairwise)
- \( \neg L\sigma \) is **contradictory** with \( \Lambda \): \( \Lambda \) contains a variant of \( \neg L\sigma \)

Context: \( P(u,u) \quad Q(a,b) \quad \sigma = \{ x \rightarrow u, \ y \rightarrow u, \ z \rightarrow b \} \)

Clause: \( \neg P(x,y) \lor \neg Q(a,z) \)

Clause \( \sigma \): \( \neg P(x,x) \lor \neg Q(a,b) \)

Close is applicable

contradictory    contradictory
Model Evolution – Further Ingredients

- **Derivation**
  - Start with sequent $\neg \forall \vdash \text{"Input Clause Set"}$
  - Apply Split and Close derivation rules (gives tree over sequents)

- **Refutation:** Every branch ends in sequent of the form $\Lambda \vdash \bot$

- **Fairness**
  - Consider a derivation with limit context $\Lambda_\infty = \bigcup_{i \geq 0} \Lambda_i$
  - Close is not applicable to any $\Lambda_i$
  - Roughly: if some ground instance $C_\gamma$ of an input clause is falsified by $\Lambda_i$
    then there is a $j > i$ such that $\Lambda_j$ satisfies $C_\gamma$
    (this can always be achieved by applying the split rule)

- **Completeness**
  - Assume a fair derivation with limit context
  - Show that $\Lambda_\infty$ constitutes a model for the input clause set
Theories – Basic Definitions

- A **Theory** $\mathcal{T}$ is a consistent set of sentences
- Consider here **universal** theories
  (no existential quantifier in prenex normal form)
- **Def:** Clause set $\Phi$ is $\mathcal{T}$-unsatisfiable iff
  $\Phi \cup \mathcal{T}$ is unsatisfiable
- **Def:** Let $\mathcal{K}$ be a set of literals and $L$ be a literal

\[
\mathcal{K} \models_\mathcal{T} L
\]

iff $\mathcal{K} \cup \mathcal{T} \not\models L$

iff for every structure $\mathcal{A}$ and every valuation $v$:

$\mathcal{A},v \models \mathcal{K} \cup \mathcal{T}$ implies $\mathcal{A},v \models L$

**Examples**

- $\{ P(u,a), u=f(u), a=f(a) \} \models_E P(f(u),f(a))$ holds
- $\{ P(u,a), u=f(u), v=f(v) \} \models_E P(f(u),f(a))$ does not hold
**ME(\(\mathcal{T}\)) – Derivation Rules (1)**

\[
\text{T-Split} \quad \frac{\Lambda \vdash \Phi, C \lor L}{\Lambda, K\sigma \vdash \Phi, C \lor L} \quad \frac{\Lambda \vdash \Phi, C \lor L}{\Lambda, \overline{K}\sigma \vdash \Phi, C \lor L}
\]

if

1. \(\sigma\) is a \(\mathcal{T}\)-context unifier of \(C \lor L\) against \(\Lambda\) with key set \(\mathcal{K} \cup \{L\}\)
2. \(K \in \neg\mathcal{K}\)
3. neither \(K\sigma\) nor \(\neg K\sigma\) is \(\mathcal{T}\)-contradictory with \(\Lambda\)

\(\sigma\) is a **\(\mathcal{T}\)-context unifier** of clause \(L_1 \lor \ldots \lor L_v\), iff there are sets \(\mathcal{K}_1, \ldots, \mathcal{K}_n\) of variants of literals from \(\Lambda\) s.th. \(\mathcal{K}_i\sigma \models_{\mathcal{T}} \neg L_i\sigma\)

Each set \(\mathcal{K}_i \cup \{L_i\}\) is called a **key set**

**Context:** \(P(a,b)\quad u = f(u)\)

**Clause:** \(\neg P(f(a), f(x))\)

**Key set:**
\[
\{ P(a,b), \quad u = f(u), \quad v = f(v), \quad \neg P(f(a), f(x)) \}
\]

\(\sigma = \{ u \rightarrow a, \quad v \rightarrow b, \quad x \rightarrow b \}\)

**T-Split on** \(\neg (a = f(a))\)
**ME(\mathcal{T}) – Derivation Rules (1)**

\[ \text{\textit{\mathcal{T}-Split}} \]

\[ \Lambda \vdash \Phi, C \vee L \]

\[ \Lambda, K\sigma \vdash \Phi, C \vee L \quad \Lambda, \overline{K}\sigma \vdash \Phi, C \vee L \]

if

1. \( \sigma \) is a \( \mathcal{T} \) -context unifier of \( C \vee L \) against \( \Lambda \) with key set \( \mathcal{K} \cup \{ L \} \)
2. \( K \in \neg \mathcal{K} \)
3. neither \( K\sigma \) nor \( \neg K\sigma \) is \( \mathcal{T} \) -contradictory with \( \Lambda \)

**\( K\sigma \) is \( \mathcal{T} \) – contradictory with \( \Lambda \)**

iff there is a set \( \mathcal{K} \) of variants of literals from \( \Lambda \) s.th. \( \mathcal{K} \models_{\mathcal{T}} \neg K_1\sigma \)

Example for \( \mathcal{T} \) – contradictory:

Context: \( P(u,v) \quad u \equiv f(u) \quad K\sigma: \neg P(f(u),f(v)) \)

\[ \mathcal{K} = \{ P(u,v), \ u \equiv f(u), \ v \equiv f(v) \} \]
ME(\mathcal{T}) – Derivation Rules (2)

\textbf{\textit{\mathcal{T}}-Repair} \quad \frac{\Lambda \vdash \Phi, C \lor L}{\Lambda, K\sigma \vdash \Phi, C \lor L}

if

1. \(\sigma\) is a \(\mathcal{T}\)-context unifier of \(C \lor L\) against \(\Lambda\) with key set \(K \cup \{ L \}\)
2. \(K \in \neg\mathcal{K}\)
3. \(K\sigma\) is not \(\mathcal{T}\)-contradictory with \(\Lambda\), but
   \(\neg K\sigma\) is \(\mathcal{T}\)-contradictory with \(\Lambda\)
4. \(\Lambda\) does not contain a variant of \(K\sigma\)

- \(\mathcal{T}\)-Repair is the one-armed, disjoint variant of \(\mathcal{T}\)-Split
- \(\mathcal{T}\)-Repair is not applicable if \(\mathcal{T}\) is the „empty“ theory

Context: \(\neg(f(a) = b)\) \quad a = b \quad P(a) \quad f(u) = u

Clause: \(\neg P(f(a))\)

\(\mathcal{T}\)-Repair with \(\neg(a = f(a))\)
ME(\mathcal{T}) – Derivation Rules (3)

\textbf{\(\mathcal{T}\)-Close} \quad \frac{\Lambda \vdash \Phi, C}{\Lambda \vdash \bot}

if (1) \(\Phi \neq \emptyset\) or \(C \neq \bot\)

(2) there is a \(\mathcal{T}\)-context unifier \(\sigma\) of \(C\) against \(\Lambda\) such that each literal of \(C\sigma\) is \(\mathcal{T}\)-contradictory with \(\Lambda\)

\textbf{Note:} Condition (2) must be decidable!
Interpretation Associated to a Context

- Crucial to understand the working of the calculus
- Basis of the completeness proof
- Basis of feasible instantiation with theory reasoners
  E.g. Waldmeister for the theory of equality
Interpretation Associated to a Context

Literal set $\mathcal{KT}$—produces a literal $L$ in $\Lambda$

Literal set $\mathcal{K}$: \{ $K_1$ ... $K_n$ \}

Instances: \{ $K_1 \gamma$ ... $K_n \gamma$ \}

Theory Reasoning: \{ $K_1 \gamma$ ... $K_n \gamma$ \} $\models_\mathcal{T} L$

No literals $L_i \in \Lambda$ such that $K_i \succ \sim L_i \succ K_i \gamma$

"$K_i$ produces $K_i \gamma$ in $\Lambda$"

Interpretation Associated to $\Lambda$

A ground atom $A$ is assigned true in $\Lambda$ via $\mathcal{K}$

iff some set $\mathcal{K}$ of variants of literals from $\Lambda \mathcal{T}$—produces $A$
Interpretation Associated to a Context

Context $\land \mathcal{T}$ – produces a literal $L$

Literal set $\mathcal{K}$ (as above): $K_1 \ldots K_n$

Instances: $K_1\gamma \ldots K_n\gamma$

Theory Reasoning: $K_1\gamma \ldots K_n\gamma \models_{\mathcal{T}} L$

Examples

$\{P(a), \ f(x)=x, \ \neg(f(a)=a)\}$ does not E-produce $P(f(a))$

$\Rightarrow P(f(a))$ is assigned false in associated E-interpretation

$\{P(a), \ f(x)=x, \ \neg P(f(a))\}$ E-produces $P(f(a))$ and $\neg P(f(a))$

$\Rightarrow P(f(a))$ is assigned true in associated E-interpretation
ME(𝛇) Calculus – Theory Reasoner $R_\mathcal{T}$

- A lifting lemma cannot be proven “once and for all“, replace it by **admissibility condition** of theory reasoner $R_\mathcal{T}$

- **Theory reasoner** $R_\mathcal{T}$
  - **Input**: a context $\Lambda$ and a clause $C = L_1 \not\in \mathcal{C} \ldots \not\in \mathcal{C} L_n$
  - **Output**: a $n+1$-tuple $(\mathcal{K}_1, \ldots, \mathcal{K}_n, \sigma)$ or undefined
  
  where $\mathcal{K}_i$ is a set of variants of literals from $\Lambda$ and $\sigma$ is a substitution

- $R_\mathcal{T}$ is **sound** iff $\mathcal{K}_i\sigma \models_\mathcal{T} \neg L_i\sigma$ (i.e. $\sigma$ is a $\mathcal{T}$–context unifier)

- $R_\mathcal{T}$ is **complete** iff the following holds:
  For every ground instance $C_\gamma$ and all sets $\mathcal{K}_1, \ldots, \mathcal{K}_n$ (as above):

  If $C_\gamma$ is assigned false in $\Lambda$ via $\mathcal{K}_1, \ldots, \mathcal{K}_n$
  
  then $R_\mathcal{T}(\Lambda, C) = (\mathcal{K}_1, \ldots, \mathcal{K}_n, \sigma)$ for some substitution $\sigma \triangleright \gamma$

- $R_\mathcal{T}$ is **admissible** iff it is sound and complete
Consequences and Properties

- Associated interpretation should be **total**: easy, context contains $\neg v$
- Associated interpretation should be a $\mathcal{T}$–interpretation

Need further restrictions on allowed theories to guarantee this:

- Non-negative theories: not $\models \exists (A_1 \land \cdots \land A_n)$
- $\mathcal{T} = \{ \neg A \}$ is not allowed

- Theory must be ground convex:
  $\models_{\mathcal{T}} B \rightarrow A_1 \lor \cdots \lor A_n$ implies $\models_{\mathcal{T}} B \rightarrow A_i$ for some $i$
  ($B$ conjunction of ground atoms, $A$ ground atom)
  $\mathcal{T} = \{ A \lor B \}$ is not allowed

- **Property**
  If limit context $\Lambda_\infty$ assigns false to a (ground) clause $C_\gamma$ via $\mathcal{K}_1, \ldots, \mathcal{K}_n$
  then
  there is an $i$ such that for all $j > i$ $\Lambda_j$ assigns false to $C_\gamma$ via $\mathcal{K}_1, \ldots, \mathcal{K}_n$

- **Completeness**
  Fairness + admissible theory reasoner will detect this situation eventually and invalidate it
Equality and Waldmeister

- **Problem**
  Waldmeister is a theorem prover for unit clauses
  \{ s_1 = t_1, \ldots, s_n = t_n, \neg (s = t) \}

  How to match it to contexts and arbitrary clauses?
  \neg (s_1 = t_1) \lor \ldots \lor \neg (s_m = t_m) \lor s_{m+1} = t_{m+1} \lor \ldots \lor s_n = t_n

- **Context Problem**
  \( \Lambda = \{ a = f(a), P(u), \neg P(a), \neg P(f(a)), \neg P(f(f(a))) \} \)
  Clause \( \neg P(a) \)

  Waldmeister has to discover instances \( P(f(f(f(a)))) \), …

  **Solution (?)**
  Convert context to equivalent set of atoms
  E.g. for signature \{a/0, b/0, f/1\} obtain
  \( \Lambda = \{ a = f(a), P(b), P(f(b)), P(f(f(b))), P(f(f(f(x)))) \} \)
  Resulting set can be infinite in case of non-linear literals!
Equality and Waldmeister

• **Problem**

Waldmeister is a theorem prover for unit clauses
\{ s_1 = t_1, \ldots, s_n = t_n, \neg(s = t) \}

How to match it to **contexts** and **arbitrary clauses**
\neg(s_1 = t_1) \lor \ldots \lor \neg(s_m = t_m) \lor s_{m+1} = t_{m+1} \lor \ldots \lor s_n = t_n

• **Arbitrary Clauses Problem**

From definition of associated interpretation it follows:

Context \Lambda falsifies a positive literal \( A \)
iff some negative literal \( \neg B \in \Lambda \) produces \( \neg A \) in \( \Lambda \)

**Consequently:**

Can resolve away positive clause literals against context literals
Leaves only rest clause \((\neg(s_1 = t_1) \lor \ldots \lor \neg(s_m = t_m))\sigma\)
Equality and Waldmeister

- **Problem**
  Waldmeister is a theorem prover for unit clauses
  \[\{ s_1=t_1, \ldots, s_n=t_n, \neg(s=t) \}\]
  How to match it to **contexts** and **arbitrary clauses**
  \[\neg(s_1=t_1) \lor \ldots \lor \neg(s_m=t_m) \lor s_{m+1}=t_{m+1} \lor \ldots \lor s_n=t_n\]

- **Arbitrary Clauses Problem**
  How to treat rest clause \(\neg(s_1=t_1) \lor \ldots \lor \neg(s_m=t_m))\sigma\)?

**Solution**

Code it as a negative unit clause (due to Thomas Hillenbrand):
\[\neg(\text{clause}(s_1,t_1,\ldots,s_m,t_m) = \text{true})\]
\[\text{clause}(x_1,x_1,\ldots,x_m,x_m) = \text{true}\]
Can easily query Waldmeister with many clauses simultaneously

- **Thus have transformation for Waldmeister now**
  But Waldmeister still has to be modified to compute „all“ solutions!
Conclusion

• Presented simplified calculus, without universal variables e.g. $\forall x \, P(x,u)$
  - Universal variables crucial for performance
    • calculus instantiates to postive hyper-resolution for Horn case
    • One call to Waldmeister for unit theories
  - Should work out without greater difficulties

• Is this all feasible?

• Difference to Ganzinger/Korovin Calculus wrt. theory reasoning
  - Works for arbitrary universal non-negative convex theories
  - Does not need a term ordering
    But using term orderings might be advantageous…