A Sound and Complete Algorithm for Simple Conceptual Logic Programs

Cristina Feier and Stijn Heymans

Institute of Information Systems, Knowledge-Based Systems Group, Vienna University of Technology

12 December 2008
Overview

- Open Answer Set Programming - Motivation, Decidable Fragments
- (Simple) Conceptual Logic Programs - Definition, Properties
- Reasoning with Simple Conceptual Logic Programs
  - Completion structure
  - Rules for evolving a completion structure
  - Termination, Soundness, Completeness

- Conclusions

- Future work
Part I

Simple Conceptual Logic Programs
Closed-World Reasoning in Answer Set Programming

\[
\begin{align*}
&\text{fail}(X) \leftarrow \neg \text{pass}(X) \\
&\text{pass}(\text{john}) \leftarrow \\
\end{align*}
\]

→ *ground* the program with all constants (*john*):

\[
\begin{align*}
&\text{fail}(\text{john}) \leftarrow \neg \text{pass}(\text{john}) \\
&\text{pass}(\text{john}) \leftarrow \\
\end{align*}
\]

→ answer set: \{\text{pass}(\text{john})\}.
Answer set: \{ \textit{pass}(john) \}.

No \textit{fail}-atom: the \textit{fail}-predicate is not satisfiable.

In the context of conceptual reasoning this is not feasible: other \textit{data} make \textit{fail} satisfiable, i.e., the program \textit{makes sense} but one is forced to introduce all significant constants.

Do not assume all possible constants are present: assume the presence of anonymous objects – \textit{open domains}.
An open answer set of $P$ is a pair $(U, M)$ where
- the universe $U$ is a non-empty superset of the constants in $P$, and
- $M$ is an answer set of $P_U$.

Examples:
- $(\{john, x\}, \{\text{pass}(john), \text{fail}(x)\})$ is open answer set since
  $\{\text{pass}(john), \text{fail}(x)\}$ is an answer set of
  
  \[
  \begin{align*}
  \text{fail}(x) & \leftarrow \neg \text{pass}(x) \\
  \text{fail}(john) & \leftarrow \neg \text{pass}(john) \\
  \text{pass}(john) & \leftarrow
  \end{align*}
  \]

- $(\{john, x_1, x_2, \ldots\}, \{\text{pass}(john), \text{fail}(x_1), \text{fail}(x_2), \ldots\})$,
- $(\{john\}, \{\text{pass}(john)\})$. 


Undecidability of Open ASP

→ shown by reduction from undecidable *domino problem*. 
Regaining Decidability

Retain openness, but restrict the shape of logic programs in order to obtain decidability.

Three types of restrictions:

- Conceptual Logic Programs
- Local Forest Logic Programs (and variations)
- Guarded Programs (and variations)
Conceptual Logic Programs

Satisfiability checking w.r.t. Conceptual Logic Programs is decidable and in \text{EXPTIME}(\text{reduction of decidability of satisfiability checking to checking non-emptiness of two-way alternating tree automata (2ATA)}).

- Only unary and binary predicates allowed: $a(X)$ and $f(X, Y)$.
- No constants.
- Four types of rules:
  - Free rules
  - Unary rules
  - Binary rules
  - Constraints

Conceptual Logic Programs have the tree model property.
CoLP Rules

Free Rules:
\[ a(X) \lor \neg a(X) \leftarrow \text{or } f(X, Y) \lor \neg f(X, Y) \leftarrow \]
→ allow for the 'free' introduction of unary and binary literals, provided other rules do not impose extra constraints.

Unary Rules:
\[ a(X) \leftarrow f(X, Y_1), \neg g(X, Y_2), h(X, Y_2), Y_1 \neq Y_2 \]
→ branching or tree structure.
→ positive connection between each node \( X \) and a successor \( Y_i \).

Binary Rules:
\[ f(X, Y) \leftarrow a(X), \neg b(X), g(X, Y), c(Y) \]

Constraints:
\[ a(X) \text{ or } f(X, Y) \]
Simple Conceptual Logic Programs: Preliminaries

- a CoLP program $P$: a set of rules
- $P_q$: the rules of $P$ (unary or binary) that have $q$ as a head predicate
- $upreds(P)$: the set of unary predicates of $P$
- $bpreds(P)$: the set of binary predicates of $P$
- $\pm p$ denotes $p$ or not $p$; $\mp p = not p$ if $\pm p = p$ and $\mp p = p$ if $\pm p = not p$
- marked predicate dependency graph of $P$:
  - nodes: $upreds(P) \cup bpreds(P)$
  - edges: $\{(p, q) \mid \exists \alpha(X) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P \lor \alpha(X, Y) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P \text{ s.t. } p \in \alpha \land q \in \beta^+ \cup \delta^+ \cup \gamma^+\}$
  - marked edges: $\{(p, q) \mid \exists r: \alpha(X) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P \lor r: \alpha(X, Y) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P \text{ s.t. } p \in \alpha \land q \in \gamma^+\}$
  - marked cycle: cycle which contains a marked edge
Simple Conceptual Logic Programs

Simple Conceptual Logic Programs differ from Conceptual Logic Programs by not allowing:

- inequalities in unary rules
- marked cycles in the marked predicate dependency graph of $P$
- constraints, although these can be simulated:

\[ \text{const} (x) \leftarrow \text{not const} (x), \text{body}, \]

for a new predicate $\text{const}$
Satisfiability checking w.r.t simple CoLPs is $\text{EXPTIME}$-complete.

Simple CoLPs have the tree model property.
Part II

An Algorithm for Simple Conceptual Logic Programs
Preliminaries—Tree notation

- Concatenation of a number \( c \in \mathbb{N}_0 \) to \( x \), where \( x \) is a sequence of numbers from \( \mathbb{N}_0 \): \( x \cdot c \), or \( xc \)
- A (finite) tree \( T \): a (finite) set of nodes, where each node is a sequence of numbers from \( \mathbb{N}_0 \) such that if \( x \cdot c \in T \) and \( c \in \mathbb{N}_0 \), then \( x \in T \);
- The empty word \( \varepsilon \) is the root of \( T \)
- \( \text{succ}_T(x) = \{x \cdot c \in T \mid c \in \mathbb{N}_0\} \): successors of \( x \)
- \( A_T = \{(x, y) \mid x, y \in T, \exists c \in \mathbb{N}_0 : y = x \cdot c\} \): the set of edges of \( T \)
- For \( x, y \in T \), \( x \leq y \) iff \( x \) is a prefix of \( y \)
- \( \text{path}_T(x, y) \): a finite path in \( T \) with \( x \) the smallest element w.r.t. the order relation \( < \) and \( y \) the greatest element
- \( T[x] \): subtree of \( T \) at \( x \);
A completion structure for a simple CoLP $P$ is a tuple: $\langle T, G, \text{ct}, \text{st}, \text{sg}, \text{nj}_U \rangle$

- $T$ is a tree - the potential universe
- $G = \langle V, E \rangle$ is a directed graph with nodes $V \subseteq \mathcal{B}_{PT}$ and edges $E \subseteq \mathcal{B}_{PT} \times \mathcal{B}_{PT}$
- $\text{ct}, \text{st}, \text{sg},$ and $\text{nj}_U$ are additional labeling functions
Labeling functions

- **content function:** $c_T : T \cup A_T \rightarrow \text{preds}(P) \cup \text{not}(\text{preds}(P))$

- **status function:**
  
  $s_T : \{(x, \pm q) \mid \pm q \in c_T(x), x \in T \cup A_T\} \rightarrow \{\text{exp}, \text{unexp}\}$

- **segment function:**
  
  $s_G : \{(x, q, r) \mid x \in T, \text{not} q \in c_T(x), r \in P_q\} \rightarrow \mathbb{N}$

- **negative justification unary function:**
  
  $n_{JU} : \{(x, q, r) \mid x \in T, \text{not} q \in c_T(x), r \in P_q\} \rightarrow 2^T$
An initial completion structure for checking the satisfiability of a unary predicate $p$ w.r.t. a simple CoLP $P$ is a completion structure with $T = \{\varepsilon\}$, $V = \{p(\varepsilon)\}$, $E = \emptyset$, and $ct(\varepsilon) = \{p\}$, $st(\varepsilon, p) = \text{unexp}$, and the other labeling functions are undefined for every input.
Initial Completion Structure - Example

\[ r_1 : \quad \text{restore}(X) \quad \leftarrow \quad \text{crash}(X), y(X, Y), \text{backSucc}(Y) \]
\[ r_2 : \quad \text{backSucc}(X) \quad \leftarrow \quad \text{not crash}(X), y(X, Y), \text{not backFail}(Y) \]
\[ r_3 : \quad \text{backFail}(X) \quad \leftarrow \quad \text{not backSucc}(X) \]
\[ r_4 : \quad \text{yesterday}(X, Y) \lor \text{not yesterday}(X, Y) \quad \leftarrow \]
\[ r_5 : \quad \text{crash}(X) \lor \text{not crash}(X) \quad \leftarrow \]

\[ \varepsilon \{ \text{restore}^{unexp} \} \]

\[ \text{CT}(\varepsilon) \]
\[ \text{ST}(\varepsilon) \]

**Figure:** Initial completion structure for restore w.r.t. \( P \)
Expansion rules

Rules which motivate the presence/absence of an atom in an open answer set. The open answer set is constructed in a top-down manner.

update\( (l, \pm p, z) \) - common operation used whenever the expansion of \( l \) leads to \( \pm p(z) \)

- if \( \pm p \notin CT(z) \), then \( CT(z) = CT(z) \cup \{ \pm p \} \) and \( ST(z, \pm p) = unexp \)
- if \( \pm p = p \) and \( \pm p(z) \notin V \), then \( V = V \cup \{ \pm p(x) \} \)
- if \( l \in B_{PT} \) and \( \pm p = p \), then \( E = E \cup \{(l, \pm p(z))\} \)
Expand unary positive

Prerequisites:
- \( p \in \mathcal{CT}(x) \) and \( \mathcal{ST}(x, p) = \text{unexp} \)

Actions:
- choose a rule which defines \( p \):
  \[
  p(x) \leftarrow \beta(x), (\gamma_m(x, y_m), \delta_m(y_m))_{1 \leq m \leq k}
  \]
- \( \text{update}(p(x), \beta, x) \)
- for each \( m, 1 \leq m \leq k \):
  - nondeterministically choose a \( y \in \text{succ}_T(x) \) or let \( y = x \cdot s \) be a new successor of \( x \)
  - \( \text{update}(p(x), \gamma_m, (x, y)) \)
  - \( \text{update}(p(x), \delta_m, y) \)
Expand unary positive - example

\[ r_1 : \quad \text{restore}(X) \leftarrow \text{crash}(X), y(X, Y), \text{backSucc}(Y) \]

**Figure:** Expansion of a unary positive literal
Choose a unary literal

Prerequisites:

- there is an $x \in T$ for which none of $\pm a \in \text{ct}(x)$ can be expanded and for all $(x, y) \in A_T$, none of $\pm f \in \text{ct}(x, y)$ can be expanded
- there is a $p \in \text{upreds}(P)$ such that $p \notin \text{ct}(x)$ and $\text{not } p \notin \text{ct}(x)$

Actions:

- add $p$ to $\text{ct}(x)$ with $\text{st}(x, p) = \text{unexp}$ or add $\text{not } p$ to $\text{ct}(x)$ with $\text{st}(x, \text{not } p) = \text{unexp}$
Choose a unary predicate - Example

$$\epsilon \{ a \} \{ restore_{r1}^{exp} \rightarrow crash^{unexp} \ ? backSucc \ ? backFail \}$$

$$\{ yesterday^{unexp} \}$$

$$1 \{ b \} \{ backSucc^{unexp} \}$$

Figure: Choose a unary predicate
Choose a unary predicate - Example

\[ \epsilon \{ a \}, \{ \text{restore}_{r1}^{\text{exp}} \} \rightarrow \text{crash}^{\text{unexp}}, \not\text{backSucc}^{\text{unexp}} \{ \}
\]

\[ \{ \text{yesterday}^{\text{unexp}} \}\]

\[ 1 \{ b \}, \{ \text{backSucc}^{\text{unexp}} \} \]

**Figure:** Choose a unary predicate
Expand unary negative

**Prerequisites (1):**
- \( \text{not } p \in \text{CT}(x) \) and \( \text{ST}(x, \text{not } p) = \text{unexp} \)

**Actions (1):**
- for every rule which defines \( p \) choose a segment \( m, 0 \leq m \leq k \):
  \[ \text{SG}(x, p, r) = m \]
  - \( m = 0 \): choose a \( \pm a \in \beta \), and update(\( \text{not } p(x), \mp a, x \)),
    \[ \text{NJ}_U(x, p, r) = \{x\}. \text{ (local justification)} \]
  - \( m > 0 \): for every \( y \in \text{succ}_T(x) \): (\( \dagger \)) choose a \( \pm a_y \in \gamma_m \cup \delta_m \),
    update(\( \text{not } p(x), \mp a_y, (x, y) \))/update(\( \text{not } p(x), \mp a_y, y \)) and
    \[ \text{NJ}_U(x, p, r) = \text{NJ}_U(x, p, r) \cup \{y\} \text{ (external justification)} \]
- \( \text{ST}(x, \text{not } p) = \text{exp} \)

**OR**

**Prerequisites (2):**
- \( \text{ST}(x, \text{not } p) = \text{exp} \) and for some \( r \in P_p, \text{SG}(x, p, r) \neq 0 \), and
  \[ \text{NJ}_U(x, p, r) = S \text{ with } |S| < |\text{succ}_T(x)| \]

**Actions (2):**
- For every \( r \) s.t. \( \text{SG}(x, p, r) = m \neq 0 \) and for every \( y \in \text{succ}_T(x) \) s.t.
  \( y \notin \text{NJ}_U(x, p, r) \): (\( \dagger \))
Expanding unary negative - local justification

\[ \epsilon \{ a \} \{ \text{restore}^{\text{exp}}_{r_1} \} \rightarrow \text{crash}^{\text{unexp}} \leftarrow \text{not backSucc}^{\text{exp}}_{\{(r_2,0,0)\}} \{ \text{yesterday}^{\text{unexp}} \} \]

Figure: Expansion of a unary negative predicate symbol
Expanding unary negative - external justification

$$r_1 : \ a(X) \leftarrow f(X, Y), b(Y), g(X, Z), d(Z)$$

Figure: Expanding unary negative: example 2
Expanding unary negative - example 2

\[ r_1 : \quad a(X) \leftarrow f(X, Y), b(Y), g(X, Z), b(Z) \]

\[
\begin{array}{c}
\times \{\text{not } a, \ldots\} \\
\{\ldots\} & \{\text{not } f, \ldots\}
\end{array}
\]

\[
\begin{array}{ccc}
x_1 \{\text{not } b, \ldots\} & x_2 \{\ldots\} & x_3 \{\ldots\}
\end{array}
\]

**Figure:** Expanding unary negative: OK
Expanding unary negative - example 2

\[ r_1 : \quad a(X) \leftarrow f(X, Y), b(Y), g(X, Z), b(Z) \]

Figure: Expanding unary negative: NOT OK
Expanding unary negative - example 2

\[ r_1 : \ a(X) \leftarrow f(X, Y), b(Y), g(X, Z), b(Z) \]

Figure: Expanding unary negative: NOT OK
Expansion rules for binary literals

Expand binary positive
- similar to expand unary positive (no need to introduce successors)

Expand binary negative
- similar to expand unary negative (the local case)

Choose binary
- similar to choose unary
Saturation

A node $x \in T$ is saturated iff:

- for all $p \in \text{upreds}(P)$, $p \in \text{ct}(x)$ or not $p \in \text{ct}(x)$ and none of $\pm a \in \text{ct}(x)$ can be further expanded
- for all $(x, y) \in A_T$ and $p \in \text{bpreds}(P)$, $p \in \text{ct}(x, y)$ or not $p \in \text{ct}(x, y)$ and none of $\pm f \in \text{ct}(x, y)$ can be further expanded

No expansions can be performed on a node from $T$ until its predecessor is saturated.
A node $x \in T$ is *blocked* iff:

- there is an ancestor $y$ of $x$ such that $c_T(x) \subseteq c_T(y)$

$x$ and $y$ as above form a blocking pair: $(x, y) \in blocked(T)$.

A blocked node is not further expanded.
Revisiting the restore example - blocking

$\epsilon \{ restore \rightarrow crash \leftarrow not \ backSucc \leftarrow backFail \}$

$\{yesterday\}$

$1 \{ not \ restore \rightarrow not \ crash \leftarrow backSucc \leftarrow not \ backFail \}$

$\{yesterday\}$

$11 \{ not \ backFail \}$

**Figure:** Blocking nodes: content equivalence
Cached nodes

A node $x \in T$ is cached iff:

- there is a saturated node $y \in T$, $y \nless x$, $x \nless y$, such that $\mathcal{CT}(x) \subseteq \mathcal{CT}(y)$

$x$ and $y$ as above are called a caching pair: $(x, y) \in \text{cached}(T)$.

No expansions can be performed on a cached node.
Contradictory, complete, clash-free completion structures

Contradictory completion structure:
- for some $x \in T$ and $a \in upreds(P)$, $\{a, \text{not } a\} \subseteq CT(x)$
- or
- for some $(x, y) \in A_T$ and $f \in bpreds(P)$, $\{f, \text{not } f\} \subseteq CT(x, y)$

Complete completion structure: a completion structure to which no rule can be further applied

Clash-free completion structure:
- it is not contradictory
- $G$ does not contain positive cycles.
A predicate symbol $p$ is satisfiable w.r.t. a Simple Conceptual Logic Program $P$ iff there is a clash-free complete completion structure for $p$ w.r.t. $P$.
Termination

Let $P$ be a simple CoLP and $p \in \text{upreds}(P)$. Then, one can construct a finite complete completion structure by a finite number of applications of the expansion rules to the initial completion structure for $p$ and $P$, taking into account the applicability rules.

Proof Sketch.

- finite number of values for $ct(x) \implies$ eventually across every branch will exist $x, y$, s.t. $ct(x) = ct(y) \implies$ blocking situation
- finite number of branches
Soundness

Let $P$ be a simple CoLP and $p \in \text{upreds}(P)$. If there exists a clash-free complete completion structure for $p$ w.r.t. $P$, then $p$ is satisfiable w.r.t. $P$.

Proof Sketch.

Construction of an OAS from a clash-free complete completion structure:

- construction of an open interpretation $(U, M)$ and of a graph $G_{\text{ext}}$ which extends $G$:
  - for every blocking or caching pair $(x, y)$: mirror the connections and the content of $x$ in $y$ or replace $T[y]$ with $T[x]$

- proof that $M$ is a minimal model of $P^M_U$
  - $M$ is a model: from the expansion rules
  - $M$ is minimal - derives from the fact that there are no cycles/infinite length paths in $G_{\text{ext}}$
Completeness

Let $P$ be a simple CoLP and $p \in \text{upreds}(P)$. If $p$ is satisfiable w.r.t. $P$, then there exists a clash-free complete completion structure for $p$ w.r.t. $P$.

**Proof Sketch.** Construction of a clash-free complete completion structure for $p$ w.r.t. $P$ starting from a tree-shaped OAS $(U, M)$ which satisfies $p$:

1. start with an initial completion structure for $p$ w.r.t. $P$ and guide the nondeterministic application of the expansion rules by $(U, M)$
2. take into account the constraints imposed by the saturation, blocking, caching, and clash rules:
   - (2.1) blocking pair $(x, y)$: cut the tree at $y$
   - (2.2) caching pair $(x, y)$: cut the tree at $y$
The algorithm runs in $\text{NEXPTIME}$, a nondeterministic level higher than the worst-case complexity characterization.

**Proof Sketch.**

- Let $CS$ be a complete completion structure.
- $CS'$ obtained from $CS$ by deleting all nodes $y$, where there is an $x$ for which $(x, y)$ is a blocking, or caching pair has at most $2^p$ nodes, $p = \| upreds(P) \|$
- $CS$ has at most $2^p(k + 1)$ nodes, $k$ - the maximal branching factor
Conclusions

- Simple CoLPS - hybrid language: combines features of LP and DL; one can simulate $\mathcal{ALCH}$
- Tableau-like algorithm
- Minimality makes blocking harder: restrictions on the language or special devices to tackle it
- Saturation of the nodes is needed in order to ensure consistency
Future Work

- Variable inequalities in rule bodies
- Allowing for constants
- Allowing for full cyclicity
Questions

...