# 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

$$\begin{aligned} \mathit{fail}(X) &\leftarrow \mathit{not}\ \mathit{pass}(X) \\ \mathit{pass}(\mathit{john}) &\leftarrow \end{aligned}$$

$\longrightarrow$ *ground* the program with all constants (*john*):

$$\begin{aligned} \mathit{fail}(\mathit{john}) &\leftarrow \mathit{not}\ \mathit{pass}(\mathit{john}) \\ \mathit{pass}(\mathit{john}) &\leftarrow \end{aligned}$$

$\longrightarrow$ answer set: $\{\mathit{pass}(\mathit{john})\}$.

# Closed-World Reasoning in Answer Set Programming (2)

- Answer set: $\{pass(john)\}$.
- No *fail*-atom: the *fail*-predicate is not satisfiable.
- In the context of conceptual reasoning this is not feasible: other *data* make *fail* satisfiable, i.e., the program *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 – *open domains*.

# Open Answer Set Programming

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:

- $(\{john, x\}, \{pass(john), fail(x)\})$ is open answer set since $\{pass(john), fail(x)\}$ is an answer set of

$$\begin{array}{rcl} fail(x) & \leftarrow & not\ pass(x) \\ fail(john) & \leftarrow & not\ pass(john) \\ pass(john) & \leftarrow & \end{array}$$

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

# Undecidability of Open ASP

$\longrightarrow$ 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 EXPTIME(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*:
$a(X) \vee not\ a(X) \leftarrow$ or $f(X,Y) \vee not\ f(X,Y) \leftarrow$
$\rightarrow$ allow for the 'free' introduction of unary and binary literals, provided other rules do not impose extra constraints.

*Unary Rules*:
$a(X) \leftarrow f(X, Y_1), not\ g(X, Y_2), h(X, Y_2), Y_1 \neq Y_2$
$\rightarrow$ *branching* or *tree* structure.
$\rightarrow$ positive connection between each *node* $X$ and a successor $Y_i$.

*Binary Rules*:
$f(X,Y) \leftarrow a(X), not\ b(X), g(X,Y), c(Y)$

*Constraints*:
$\leftarrow a(X)$ or $\leftarrow f(X,Y)$

# 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: $\{(p, q) \mid \exists \alpha(X) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P \vee \alpha(X, Y) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P$ s.t. $p \in \alpha \wedge q \in \beta^+ \cup \delta^+ \cup \gamma^+\}$
  - marked edges:$\{(p, q) \mid \exists r : \alpha(X) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P \vee r : \alpha(X, Y) \leftarrow \beta(X), \delta(X, Y), \gamma(Y) \in P$ s.t. $p \in \alpha \wedge q \in \gamma^+\}$
  - 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:

  $$\leftarrow body$$

  can be replaced by the simple CoLP rule:

  $$const(x) \leftarrow not\ const(x), body,$$

  for a new predicate *const*

# Properties of Simple Conceptual Logic Programs

Satisfiability checking w.r.t simple CoLPs is 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 $T$
- $succ_T(x) = \{x \cdot c \in T \mid c \in \mathbb{N}_0\}$: 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 $y$
- $path_T(x, y)$: 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 $x$;

# Completion Structure for a Simple CoLP

A completion structure for a simple CoLP $P$ is a tuple: $\langle T, G, \text{CT}, \text{ST}, \text{SG}, \text{NJ}_{\text{U}} \rangle$

- $T$ is a tree - the potential universe
- $G = \langle V, E \rangle$ is a directed graph with nodes $V \subseteq \mathcal{B}_{P_T}$ and edges $E \subseteq \mathcal{B}_{P_T} \times \mathcal{B}_{P_T}$
- $\text{CT}$, $\text{ST}$, $\text{SG}$, and $\text{NJ}_{\text{U}}$ are additional labeling functions

# Labeling functions

- *content* function: $\textsc{ct} : T \cup A_T \to \mathit{preds}(P) \cup \mathit{not}\ (\mathit{preds}(P))$
- *status* function:
  $\textsc{st} : \{(x, \pm q) \mid \pm q \in \textsc{ct}(x), x \in T \cup A_T\} \to \{\mathit{exp}, \mathit{unexp}\}$
- *segment* function:
  $\textsc{sg} : \{(x, q, r) \mid x \in T, \mathit{not}\ q \in \textsc{ct}(x), r \in P_q\} \to \mathbb{N}$
- *negative justification unary* function:
  $\textsc{nj}_\textsc{u} : \{(x, q, r) \mid x \in T, \mathit{not}\ q \in \textsc{ct}(x), r \in P_q\} \to 2^T$

# Initial Completion Structure

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 $\textsc{ct}(\varepsilon) = \{p\}$, $\textsc{st}(\varepsilon, p) = \mathit{unexp}$, and the other labeling functions are undefined for every input

# Initial Completion Structure - Example

$$
\begin{array}{llcl}
r_1: & restore(X) & \leftarrow & crash(X), y(X,Y), backSucc(Y) \\
r_2: & backSucc(X) & \leftarrow & not\ crash(X), y(X,Y), not\ backFail(Y) \\
r_3: & backFail(X) & \leftarrow & not\ backSucc(X)
\end{array}
$$

$$
\begin{array}{lll}
r_4: & yesterday(X,Y) \vee not\ yesterday(X,Y) & \leftarrow \\
r_5: & crash(X) \vee not\ crash(X) & \leftarrow
\end{array}
$$

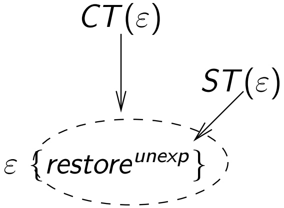


Figure: Initial completion structure for *restore* w.r.t. $P$

# 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 $\pm p \notin \mathrm{CT}(z)$, then $\mathrm{CT}(z) = \mathrm{CT}(z) \cup \{\pm p\}$ and $\mathrm{ST}(z, \pm p) = unexp$
- if $\pm p = p$ and $\pm p(z) \notin V$, then $V = V \cup \{\pm p(x)\}$
- if $l \in \mathcal{B}_{P_T}$ and $\pm p = p$, then $E = E \cup \{(l, \pm p(z))\}$

# Expand unary positive

*Prerequisites*:

- $p \in \mathrm{CT}(x)$ and $\mathrm{ST}(x, p) = unexp$

*Actions*:

- choose a rule which defines $p$:
  $p(x) \leftarrow \beta(x), \big(\gamma_m(x, y_m), \delta_m(y_m)\big)_{1 \leq m \leq k}$
- $update(p(x), \beta, x)$
- for each $m, 1 \leq m \leq k$:
  - nondeterministically choose a $y \in succ_T(x)$ or let $y = x \cdot s$ be a new successor of $x$
  - $update(p(x), \gamma_m, (x, y))$
  - $update(p(x), \delta_m, y)$

# Expand unary positive - example

$$r_1 : \quad restore(X) \quad \leftarrow \quad crash(X), y(X, Y), backSucc(Y)$$

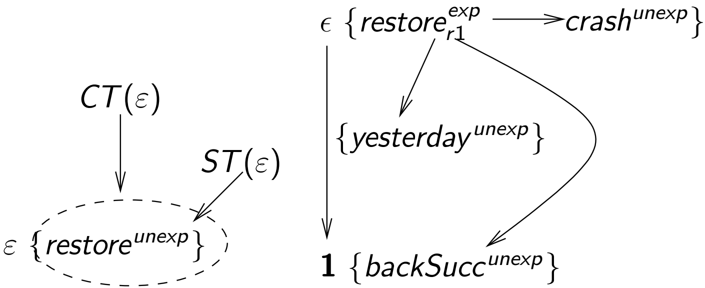


Figure: Expansion of a unary positive literal

# Choose a unary literal

*Prerequisites*:

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

*Actions*:

- add $p$ to $\mathrm{CT}(x)$ with $\mathrm{ST}(x, p) = \mathit{unexp}$ or add $\mathit{not}\ p$ to $\mathrm{CT}(x)$ with $\mathrm{ST}(x, \mathit{not}\ p) = \mathit{unexp}$

# Choose a unary predicate - Example

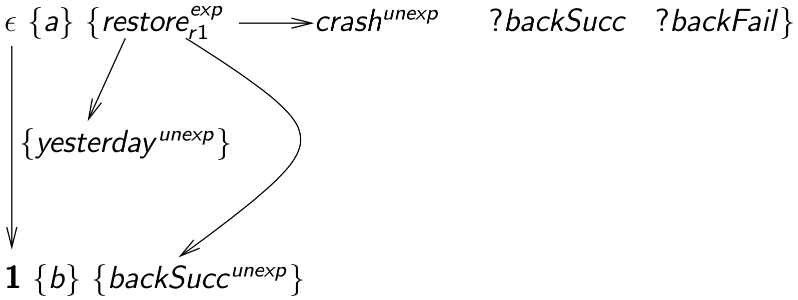


Figure: Choose a unary predicate

# Choose a unary predicate - Example

$$\begin{tikzcd}
\epsilon\ \{a\}\ \{\mathit{restore}^{exp}_{r1} \arrow[r] \arrow[d] \arrow[dd, bend left] \arrow[dd] & \mathit{crash}^{unexp} & \mathit{notbackSucc}^{unexp}\} \\
\{\mathit{yesterday}^{unexp}\} \\
\mathbf{1}\ \{b\}\ \{\mathit{backSucc}^{unexp}\}
\end{tikzcd}$$

Figure: Choose a unary predicate

# Expand unary negative

*Prerequisites (1)*:

- $not\ \ p \in \mathrm{CT}(x)$ and $\mathrm{ST}(x, not\ \ p) = unexp$

*Actions (1)*:

- for every rule which defines $p$ choose a segment $m, 0 \leq m \leq k$: $\mathrm{SG}(x, p, r) = m$
  - $m = 0$: choose a $\pm a \in \beta$, and $update(not\ \ p(x), \mp a, x)$, $\mathrm{NJ_U}(x, p, r) = \{x\}$. (*local justification*)
  - $m > 0$: for every $y \in succ_T(x)$: (†) choose a $\pm a_y \in \gamma_m \cup \delta_m$, $update(not\ \ p(x), \mp a_y, (x, y))/update(not\ \ p(x), \mp a_y, y)$ and $\mathrm{NJ_U}(x, p, r) = \mathrm{NJ_U}(x, p, r) \cup \{y\}$ (*external justification*).
- $\mathrm{ST}(x, not\ \ p) = exp$

OR

*Prerequisites (2)*:

- $\mathrm{ST}(x, not\ \ p) = exp$ and for some $r \in P_p$, $\mathrm{SG}(x, p, r) \neq 0$, and $\mathrm{NJ_U}(x, p, r) = S$ with $|S| < |succ_T(x)|$

*Actions (2)*:

- For every $r$ s.t. $\mathrm{SG}(x, p, r) = m \neq 0$ and for every $y \in succ_T(x)$ s.t. $y \notin \mathrm{NJ_U}(x, p, r)$: (†)

# Expanding unary negative - local justification

$$\begin{tikzcd}
\epsilon \arrow[dd] & \{a\}\ \{\mathit{restore}^{exp}_{r1} \arrow[r] \arrow[d] \arrow[dd, bend left] & \mathit{crash}^{unexp} & \mathit{not}\ \ \mathit{backSucc}^{exp}_{\{(r_2,0,0)\}}\} \arrow[l] \\
 & \{\mathit{yesterday}^{unexp}\} & & \\
\mathbf{1} & \{b\}\ \{\mathit{backSucc}^{unexp}\} & &
\end{tikzcd}$$

Figure: Expansion of a unary negative predicate symbol

# Expanding unary negative - external justification

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

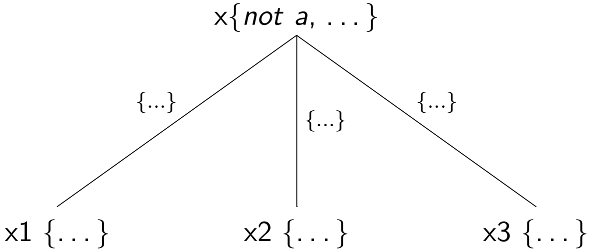


Figure: Expanding unary negative: example 2

# Expanding unary negative - example 2

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

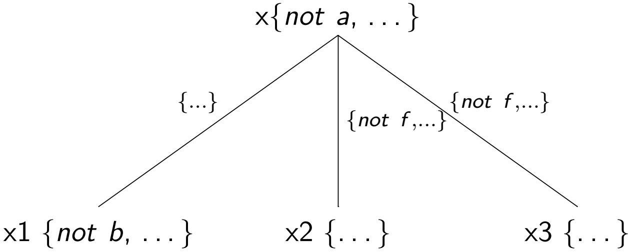


Figure: Expanding unary negative: OK

# Expanding unary negative - example 2

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

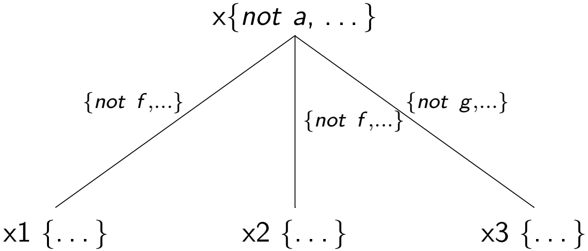


Figure: Expanding unary negative: NOT OK

# Expanding unary negative - example 2

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

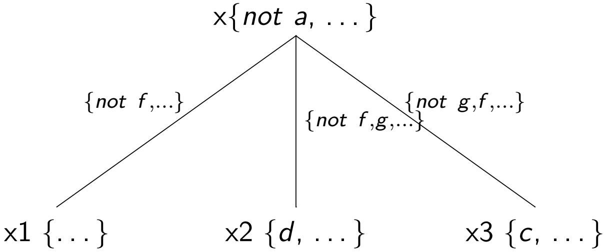


Figure: Expanding unary negative: NOT OK

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

No expansions can be performed on a node from $T$ until its predecessor is saturated.

# Blocking

A node $x \in T$ is *blocked* iff:

- there is an ancestor $y$ of $x$ such that $\textsc{ct}(x) \subseteq \textsc{ct}(y)$

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

A blocked node is not further expanded.

# Revisiting the *restore* example - blocking

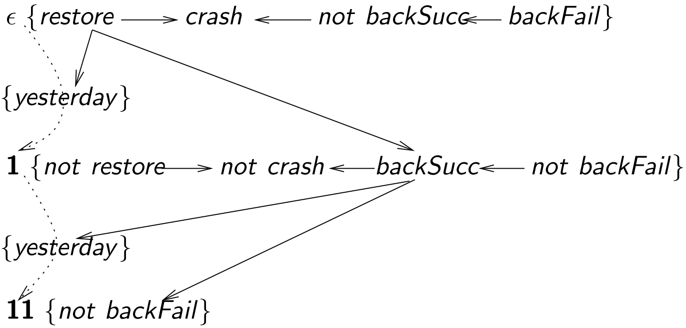


Figure: Blocking nodes: content equivalence

## Cached nodes

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

- there is a saturated node $y \in T$, $y \not\leq x$, $x \not\leq y$, such that $\mathrm{CT}(x) \subseteq \mathrm{CT}(y)$

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

No expansions can be performed on a cached node.

# Contradictory, complete, clash-free completion structures

*Contradictory* completion structure:

- for some $x \in T$ and $a \in \mathit{upreds}(P)$, $\{a, \mathit{not}\ a\} \subseteq \mathrm{CT}(x)$

or

- for some $(x, y) \in A_T$ and $f \in \mathit{bpreds}(P)$, $\{f, \mathit{not}\ f\} \subseteq \mathrm{CT}(x, y)$

*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.

# Characterization of satisfiability in terms of a completion structure

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 $p \in \mathit{upreds}(P)$. 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 $\mathrm{CT}(x) \Longrightarrow$ eventually across every branch will exist $x$, $y$, s.t. $\mathrm{CT}(x) = \mathrm{CT}(y) \Longrightarrow$ blocking situation
- finite number of branches

# Soundness

Let $P$ be a simple CoLP and $p \in upreds(P)$. 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 $G_{ext}$ 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 $P_U^M$
  - $M$ is a model: from the expansion rules
  - $M$ is minimal - derives from the fact that there are no cycles/infinite length paths in $G_{ext}$

# Completeness

Let $P$ be a simple CoLP and $p \in \mathit{upreds}(P)$. 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 $y$

# Complexity

The algorithm runs in 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

…