Feature constraints
to modelise Unix filesystems

Nicolas Jeannerod

IRIF

February 7, 2018
The CoLiS Project
The CoLiS Project

Shell \rightarrow IL

Translation
The CoLiS Project

Shell \[\rightarrow\] IL

Translation

Specification in Tree Transducers
The CoLiS Project

Shell → IL
Translation

IL → Specification in Feature Trees
IL → Specification in Tree Transducers
The CoLiS Project

Translation

Shell → IL

IL → Specification in Feature Trees

IL → Specification in Tree Transducers
Specifications.. then what?

Find accessible states that lead to errors.
Specifications.. then what?

Find accessible states that lead to errors.

▶ “Accessible”? Where the specification is satisfiable.
Specifications.. then what?

Find accessible states that lead to errors.

- “Accessible”? Where the specification is satisfiable.
- “Lead to errors”? Where the script exists abnormally.
Specifications.. then what?

Find accessible states that lead to errors.

- “Accessible”? Where the specification is satisfiable.
- “Lead to errors”? Where the script exists abnormally.

Fill automated report to script’s maintainer.
Specifications.. then what?

Find accessible states that lead to errors.

▶ “Accessible”? Where the specification is satisfiable.
▶ “Lead to errors”? Where the script exists abnormally.

Fill automated report to script’s maintainer.

Check properties
Specifications.. then what?

Find accessible states that lead to errors.

▶ “Accessible”? Where the specification is satisfiable.
▶ “Lead to errors”? Where the script exists abnormally.

Fill automated report to script’s maintainer.

Check properties:

▶ $\forall r_{in}, r_{out} \cdot (\text{spec}_{s_1}(r_{in}, r_{out}) \leftrightarrow \text{spec}_{s_2}(r_{out}, r_{in}))$
Specifications.. then what?

Find accessible states that lead to errors.
- “Accessible”? Where the specification is satisfiable.
- “Lead to errors”? Where the script exists abnormally.

Fill automated report to script’s maintainer.

Check properties:
- $\forall r_{in}, r_{out} \cdot (\text{spec}_{s_1}(r_{in}, r_{out}) \leftrightarrow \text{spec}_{s_2}(r_{out}, r_{in}))$
- $\forall r_{in}, r_{out} \cdot (\text{spec}_{s}(r_{in}, r_{out}) \rightarrow r_{out}[\text{home}] = r_{in}[\text{home}])$
Specifications.. then what?

Find accessible states that lead to errors.

▶ “Accessible”? Where the specification is satisfiable.
▶ “Lead to errors”? Where the script exists abnormally.

Fill automated report to script’s maintainer.

Check properties:

▷ ∀r_{in}, r_{out} · (spec_{s1}(r_{in}, r_{out}) ↔ spec_{s2}(r_{out}, r_{in}))
▷ ∀r_{in}, r_{out} · (spec_{s}(r_{in}, r_{out}) → r_{out}[home] = r_{in}[home])
▷ ∀r_{in}, r_{out} · (spec_{s}(r_{in}, r_{out}) ↔ r_{out} = r_{in})
Specifications.. then what?

Find accessible states that lead to errors.

▶ “Accessible”? Where the specification is satisfiable.

▶ “Lead to errors”? Where the script exists abnormally.

Fill automated report to script’s maintainer.

Check properties:

▶ $\forall r_{in}, r_{out} \cdot (\text{spec}_{s_1}(r_{in}, r_{out}) \iff \text{spec}_{s_2}(r_{out}, r_{in}))$

▶ $\forall r_{in}, r_{out} \cdot (\text{spec}_{s}(r_{in}, r_{out}) \rightarrow r_{out}[\text{home}] = r_{in}[\text{home}])$

▶ $\forall r_{in}, r_{out} \cdot (\text{spec}_{s}(r_{in}, r_{out}) \iff r_{out} \simeq r_{in})$

▶ $\forall r_{in}, r_{out} \cdot (\exists r' \cdot (\text{spec}_{s_1}(r_{in}, r') \land \text{spec}_{s_2}(r', r_{out})) \iff r_{out} \simeq r_{in})$
1. Description of filesystems
   Unix filesystems
   Static description
   Directory update

2. Constraints
   Definitions
   Basic constraints
   Negation

3. Usages
   Decidability of the First-Order Theory
   Automated Specification for Scripts: Proof of Concept
Unix filesystem

- Basically a tree with labelled nodes and edges;
Unix filesystem

- Basically a tree with labelled nodes and edges;
- There can be sharing at the leafs (hard link between files);
Unix filesystem

- Basically a tree with labelled nodes and edges;
- There can be sharing at the leafs (hard link between files);
- There can be pointers to other parts of the tree (symbolic links)
Unix filesystem

- Basically a tree with labelled nodes and edges;
- There can be sharing at the leafs (hard link between files);
- There can be pointers to other parts of the tree (symbolic links) which may form cycles.
# Table of Contents

1. Description of filesystems
   - Unix filesystems
   - Static description
   - Directory update

2. Constraints
   - Definitions
   - Basic constraints
   - Negation

3. Usages
   - Decidability of the First-Order Theory
   - Automated Specification for Scripts: Proof of Concept
\[
\exists u, v, x, w \cdot \{ \begin{align*}
& r[\text{usr}] \land v[\text{lib}] \land x[\text{ocaml}] \\
& r[\text{etc}] \land w[\text{skel}] \land u[\emptyset]
\end{align*} \}
\]
\[
\begin{align*}
&\exists u,v,x,w \cdot \\
&\quad \{ r[usr] \land v[lib] \land x[ocaml] \uparrow \land r[etc] \land w[skel] \land u[∅] \}
\end{align*}
\]
\[ r \]

\[ \{ \text{usr} \} \land \text{usr} \land \text{lib} \land \text{lib} \land \text{ocaml} \land \text{etc} \land \text{etc} \land \text{skel} \land \text{skel} \land \emptyset \land \emptyset \]

\[
\begin{align*}
c & =
\end{align*}
\]
\[ c = \exists u, v, x, w \cdot \{ \} \]
\[ c = \exists u, v, x, w \cdot \left\{ \begin{array}{l} r[\text{usr}]v \land v[\text{lib}]x \\ \land r[\text{etc}]w \land w[\text{skel}]u \end{array} \right\} \]
$c = \exists u, v, x, w \cdot \left\{ r[\text{usr}] v \land v[\text{lib}] x \land x[\text{ocaml}] \uparrow \land r[\text{etc}] w \land w[\text{skel}] u \right\}$
\[ c = \exists u, v, x, w \cdot \begin{cases} r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow \\
\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u[\emptyset] \end{cases} \]
1. Description of filesystems
   Unix filesystems
   Static description
   Directory update

2. Constraints
   Definitions
   Basic constraints
   Negation

3. Usages
   Decidability of the First-Order Theory
   Automated Specification for Scripts: Proof of Concept
Directory update

```
usr/
  etc

lib/

/ocaml
```
Directory update

```
usr/
  \etc

lib/

/ocaml

mkdir /usr/lib/ocaml
```
Directory update

```
usr/  
  |   etc
  |
lib/  
  
  /ocaml

mkdir /usr/lib/ocaml

usr/  
  |   etc
  |
lib/  
  
  /ocaml
  ∅
```
Directory update

```
usr/  etc
    /lib/
    /ocaml

mkdir /usr/lib/ocaml

usr/  etc
    /lib/
    /ocaml

∅

\[ c' = \]
```
Directory update

\[
c' = \exists v, v', x, x', y'. \left\{ \right. 
\]

mkdir /usr/lib/ocaml
Directory update

\[
\begin{align*}
\text{mkdir} /\text{usr/lib/ocaml} & \rightarrow \\
\end{align*}
\]

\[
c' = \exists v, v', x, x', y'. \begin{cases} 
  r' \text{ is } r \text{ with } \text{usr} \rightarrow v' \\
  \land v' \text{ is } v \text{ with } \text{lib} \rightarrow x' \\
  \land x' \text{ is } x \text{ with } \text{ocaml} \rightarrow y' \\
  \land y'[\emptyset]
\end{cases}
\]
Er.. is that really what we want?

▶ Asymmetric:

\[ y \text{ is } x \text{ with } f \rightarrow v \]
Er.. is that really what we want?

- Asymmetric:
  
  \[ y \text{ is } x \text{ with } f \rightarrow v \]

- Makes it hard to eliminate variables:
  
  \[ \exists x \cdot \left( y \text{ is } x \text{ with } f \rightarrow v \wedge z \text{ is } x \text{ with } g \rightarrow w \right) \]
Er.. is that really what we want?

- Asymmetric:
  
  \[
  y \text{ is } x \text{ with } f \to v
  \]

- Makes it hard to eliminate variables:
  
  \[
  \exists x \cdot \left( y \text{ is } x \text{ with } f \to v \land z \text{ is } x \text{ with } g \to w \right)
  \]

- Contains in fact two pieces of information:
Er.. is that really what we want?

▶ Asymmetric:

\[ y \text{ is } x \text{ with } f \rightarrow v \]

▶ Makes it hard to eliminate variables:

\[ \exists x \cdot \left( y \text{ is } x \text{ with } f \rightarrow v \land z \text{ is } x \text{ with } g \rightarrow w \right) \]

▶ Contains in fact two pieces of information:

▶ “\( y \) and \( x \) may be different in \( f \) but are identical everywhere else”
Er.. is that really what we want?

► Asymmetric:

\[ y \text{ is } x \text{ with } f \rightarrow v \]

► Makes it hard to eliminate variables:

\[ \exists x \cdot \left( \begin{array}{c}
y \text{ is } x \text{ with } f \rightarrow v \\
\land z \text{ is } x \text{ with } g \rightarrow w
\end{array} \right) \]

► Contains in fact two pieces of information:

► “\( y \) and \( x \) may be different in \( f \) but are identical everywhere else”

► “\( y \) points to \( v \) through \( f \)”
Er.. is that really what we want?

- Asymmetric:
  
  \[ y \text{ is } x \text{ with } f \to v \]

- Makes it hard to eliminate variables:
  
  \[ \exists x \cdot \left( y \text{ is } x \text{ with } f \to v \land z \text{ is } x \text{ with } g \to w \right) \]

- Contains in fact two pieces of information:
  
  - "\( y \) and \( x \) may be different in \( f \) but are identical everywhere else"

  - "\( y \) points to \( v \) through \( f \)":
    
    \[ y[f]v \]
Er.. is that really what we want?

▶ Asymmetric:

\[ y \text{ is } x \text{ with } f \rightarrow v \]

▶ Makes it hard to eliminate variables:

\[ \exists x \cdot \left( \begin{array}{c} y \text{ is } x \text{ with } f \rightarrow v \\ \land z \text{ is } x \text{ with } g \rightarrow w \end{array} \right) \]

▶ Contains in fact two pieces of information:
  ▶ “\( y \text{ and } x \text{ may be different in } f \text{ but are identical everywhere else} \)”:

\[ y \sim_f x \]

▶ “\( y \text{ points to } v \text{ through } f \)”:

\[ y[f]v \]
Much better

- Allows to express the update:

\[
\text{“y is x with } f \rightarrow v\text{”} := y \sim_f x \land y[f] v
\]
∼: Much better

Allows to express the update:

“y is x with f → v” := y ∼_f x ∧ y[f]v

Symmetric and transitive:

\[ y \sim_f x \iff x \sim_f y \]
\[ y \sim_f x \land z \sim_f x \implies y \sim_f z \]
∼: Much better

▸ Allows to express the update:

\[ "y \text{ is } x \text{ with } f \rightarrow v" \quad := \quad y \sim_f x \land y[f]v \]

▸ Symmetric and transitive:

\[
\begin{align*}
y \sim_f x & \iff x \sim_f y \\
y \sim_f x \land z \sim_f x & \implies y \sim_f z
\end{align*}
\]

▸ Other properties:

\[
\begin{align*}
y \sim_f x \land z \sim_g x & \implies y \sim_{\{f,g\}} z \\
y \sim_f x \land y \sim_g x & \iff y \sim_{\emptyset} x
\end{align*}
\]
\(~\) Much better

- Allows to express the update:

\[
\text{“y is x with } f \rightarrow v \text{”} \quad := \quad y \sim_f x \land y[f]v
\]

- Symmetric and transitive:

\[
y \sim_f x \iff x \sim_f y \\
y \sim_f x \land z \sim_f x \quad \implies \quad y \sim_f z
\]

- Other properties:

\[
y \sim_f x \land z \sim_g x \quad \implies \quad y \sim \{f, g\} z \\
y \sim_f x \land y \sim_g x \quad \iff \quad y \sim \emptyset x
\]

- Allows to remove variables:

\[
\exists x \cdot \left( y \text{ is x with } f \rightarrow v \land z \text{ is x with } g \rightarrow w \right)
\]
∼: Much better

- Allows to express the update:

  \[ \text{“y is x with } f \rightarrow v \text{”} \; :\! = \; y \sim_f x \land y[f]v \]

- Symmetric and transitive:

  \[
  \begin{align*}
  y \sim_f x & \iff x \sim_f y \\
  y \sim_f x \land z \sim_f x & \implies y \sim_f z
  \end{align*}
  \]

- Other properties:

  \[
  \begin{align*}
  y \sim_f x \land z \sim_g x & \implies y \sim \{f, g\} z \\
  y \sim_f x \land y \sim_g x & \iff y \sim \emptyset x
  \end{align*}
  \]

- Allows to remove variables:

  \[
  \exists x \cdot \left( y \sim_f x \land y[f]v \land z \sim_g x \land z[g]w \right)
  \]
\(\sim: \text{ Much better}\)

- Allows to express the update:

\[\text{“y is x with } f \rightarrow v\text{” } := \quad y \sim_f x \land y[f]v\]

- Symmetric and transitive:

\[
y \sim_f x \iff x \sim_f y\]
\[
y \sim_f x \land z \sim_f x \implies y \sim_f z\]

- Other properties:

\[
y \sim_f x \land z \sim_g x \implies y \sim\{f,g\} z\]
\[
y \sim_f x \land y \sim_g x \iff y \sim\emptyset x\]

- Allows to remove variables:

\[
\exists x . \left( y \sim_f x \land y[f]v \land z \sim_g x \land z[g]w \right) \iff y[f]v \land z[g]w\]
∼: Much better

- Allows to express the update:

\[ "y \text{ is } x \text{ with } f \rightarrow v" \quad :\quad y \sim_f x \land y[f]v \]

- Symmetric and transitive:

\[
y \sim_f x \iff x \sim_f y \\
y \sim_f x \land z \sim_f x \implies y \sim_f z
\]

- Other properties:

\[
y \sim_f x \land z \sim_g x \implies y \sim\{f,g\} z \\
y \sim_f x \land y \sim_g x \iff y \sim\emptyset x
\]

- Allows to remove variables:

\[
\exists x \cdot \left( y \sim_f x \land y[f]v \land z \sim_g x \land z[g]w \right) \iff y[f]v \land z[g]w \land y \sim\{f,g\} z
\]
1. Description of filesystems
   Unix filesystems
   Static description
   Directory update

2. Constraints
   Definitions
   Basic constraints
   Negation

3. Usages
   Decidability of the First-Order Theory
   Automated Specification for Scripts: Proof of Concept
Model and Constraints

\[
\text{ftree} ::= \mathcal{F} \rightsquigarrow \text{ftree}
\]
Model and Constraints

\[ \text{ftree} ::= \mathcal{F} \mapsto \text{ftree} \]

- \( \mathcal{F} \) infinite set of features (names for the edges);
- \( \mathcal{F} \mapsto \text{ftree} \): partial function with finite domain;
Model and Constraints

\[ \text{ftree} ::= \mathcal{F} \rightsquigarrow \text{ftree} \]

- \( \mathcal{F} \): infinite set of features (names for the edges);
- \( \mathcal{F} \rightsquigarrow \text{ftree} \): partial function with finite domain;
- Infinite set of variables \( x, y, \text{etc.} \);
- \( f \in \mathcal{F}, \mathcal{F} \subset \mathcal{F} \) finite.

**Equality** \( x = y \)

**Feature** \( x[f]y \) \( x[f] \uparrow \)

**Fence** \( x[F] \)

**Absence** \( x \sim_F y \)

**Similarity**
Model and Constraints

\[ \text{ftree} ::= \mathcal{F} \rightsquigarrow \text{ftree} \]

- \( \mathcal{F} \): infinite set of features (names for the edges);
- \( \mathcal{F} \rightsquigarrow \text{ftree} \): partial function with finite domain;
- Infinite set of variables \( x, y, \text{etc.} \);
- \( f \in \mathcal{F}, \mathcal{F} \subset \mathcal{F} \) finite.

Equality  \( x \overset{\cdot}{=} y \)

Feature  \( x[f]y \)  \( x[f] \uparrow \) Absence

Fence  \( x[F] \)  \( x \overset{\sim}{\sim} \mathcal{F} y \) Similarity

- Composed with \( \neg, \land, \lor, \exists x, \forall x \) (no quantification on features);
Model and Constraints

\[ \text{ftree} ::= \mathcal{F} \leadsto \text{ftree} \]

- \( \mathcal{F} \) infinite set of features (names for the edges);
- \( \mathcal{F} \leadsto \text{ftree} \): partial function with finite domain;
- Infinite set of variables \( x, y \), etc.;
- \( f \in \mathcal{F}, \ F \subset \mathcal{F} \) finite.

Equality \( x = y \)

Feature \( x[f]y \) \( x[f] \uparrow \)

Fence \( x[F] \)

Absence \( x \sim_F y \)

Similarity

- Composed with \( \neg, \land, \lor, \exists x, \forall x \) (no quantification on features);
- Wanted: (un)satisfiability of these constraints;
- Bonus point for incremental procedures.
\[ \mathcal{T}, \rho \models c \]

- \( \mathcal{T} \) the model of all feature trees;
- \( \rho : \mathcal{V}(c) \to \mathcal{T} \);
Semantics

\[ \mathcal{T}, \rho \models c \]

- \( \mathcal{T} \) the model of all feature trees;
- \( \rho : \mathcal{V}(c) \to \mathcal{T} \);

Equality: \( \mathcal{T}, \rho \models x \dot{=} y \) if \( \rho(x) = \rho(y) \)
Semantics

\[ \mathcal{T}, \rho \models c \]

- \( \mathcal{T} \) the model of all feature trees;
- \( \rho : \mathcal{V}(c) \to \mathcal{T} \);

Equality: \( \mathcal{T}, \rho \models x \equiv y \) if \( \rho(x) = \rho(y) \)

Feature: \( \mathcal{T}, \rho \models x[f]y \) if \( \rho(x)(f) = \rho(y) \)

Absence: \( \mathcal{T}, \rho \models x[f] \uparrow \) if \( f \notin \text{dom}(\rho(x)) \)
Semantics

\[ \mathcal{T}, \rho \models c \]

- \( \mathcal{T} \) the model of all feature trees;
- \( \rho : \mathcal{V}(c) \rightarrow \mathcal{T} \);

**Equality:** \( \mathcal{T}, \rho \models x \equiv y \) if \( \rho(x) = \rho(y) \)

**Feature:** \( \mathcal{T}, \rho \models x[f]y \) if \( \rho(x)(f) = \rho(y) \)

**Absence:** \( \mathcal{T}, \rho \models x[f] \uparrow \) if \( f \notin \text{dom}(\rho(x)) \)

**Fence:** \( \mathcal{T}, \rho \models x[F] \) if \( \text{dom}(\rho(x)) \subseteq F \)
\[ \mathcal{T}, \rho \models c \]

- \( \mathcal{T} \) the model of all feature trees;
- \( \rho : \mathcal{V}(c) \to \mathcal{T} \);

**Equality:** \( \mathcal{T}, \rho \models x \equiv y \) if \( \rho(x) = \rho(y) \)

**Feature:** \( \mathcal{T}, \rho \models x[f]y \) if \( \rho(x)(f) = \rho(y) \)

**Absence:** \( \mathcal{T}, \rho \models x[f] \uparrow \) if \( f \notin \text{dom}(\rho(x)) \)

**Fence:** \( \mathcal{T}, \rho \models x[F] \) if \( \text{dom}(\rho(x)) \subseteq F \)

**Similarity:** \( \mathcal{T}, \rho \models x \sim_F y \) if \( \rho(x) \upharpoonright \overline{F} = \rho(y) \upharpoonright \overline{F} \)
Table of Contents

1. Description of filesystems
   Unix filesystems
   Static description
   Directory update

2. Constraints
   Definitions
   Basic constraints
   Negation

3. Usages
   Decidability of the First-Order Theory
   Automated Specification for Scripts: Proof of Concept
Game plan

- Rewriting system;
Game plan

- Rewriting system;

- Puts constraints in normal form (not necessarily unique);
Game plan

- Rewriting system;
- Puts constraints in normal form (not necessarily unique);
- Respects equivalences;
Game plan

- Rewriting system;
- Puts constraints in normal form (not necessarily unique);
- Respects equivalences;
- Normal forms: either $\perp$ or with nice properties.
Basic rewriting system

\[ x_1[f_1]x_2 \land \ldots \land x_n[f_n]x_1 \quad (n \geq 1) \]
\[ x[f]y \land x[f] \uparrow \]
\[ x[f]y \land x[F] \quad (f \notin F) \]

Clash Patterns
Basic rewriting system

\[ x_1[f_1]x_2 \land \ldots \land x_n[f_n]x_1 \quad (n \geq 1) \]
\[ x[f]y \land x[f] \uparrow \]
\[ x[f]y \land x[F] \quad (f \notin F) \]

Clash Patterns

\[
\exists X, x \cdot (x = y \land c) \quad \Rightarrow \quad \exists X \cdot c\{x \mapsto y\} \quad (x \neq y)
\]
\[
\exists X, z \cdot (x[f]y \land x[f]z \land c) \quad \Rightarrow \quad \exists X \cdot (x[f]y \land c\{z \mapsto y\}) \quad (y \neq z)
\]
\[
x \sim_F y \land x \sim_G y \land c \quad \Rightarrow \quad x \sim_{F \cap G} y \land c
\]

Simplification Rules
Basic rewriting system

\[ x_1[f_1]x_2 \land \ldots \land x_n[f_n]x_1 \quad (n \geq 1) \]
\[ x[f]y \land x[f] \uparrow \]
\[ x[f]y \land x[F] \quad (f \notin F) \]

Clash Patterns

\[ \exists X, x \cdot (x = y \land c) \quad \Rightarrow \quad \exists X \cdot c\{x \mapsto y\} \quad (x \neq y) \]
\[ \exists X, z \cdot (x[f]y \land x[f]z \land c) \quad \Rightarrow \quad \exists X \cdot (x[f]y \land c\{z \mapsto y\}) \quad (y \neq z) \]
\[ x \sim_F y \land x \sim_G y \land c \quad \Rightarrow \quad x \sim_{F \cap G} y \land c \]

Simplification Rules

\[ x \sim_F y \land x[f]z \land c \quad \Rightarrow \quad x \sim_F y \land x[f]z \land y[f]z \land c \quad (f \notin F) \]
\[ x \sim_F y \land x[f] \uparrow \land c \quad \Rightarrow \quad x \sim_F y \land x[f] \uparrow \land y[f] \uparrow \land c \quad (f \notin F) \]
\[ x \sim_F y \land x[G] \land c \quad \Rightarrow \quad x \sim_F y \land x[G] \land y[F \cup G] \land c \]
\[ x \sim_F y \land x \sim_G z \land c \quad \Rightarrow \quad x \sim_F y \land x \sim_G z \land y \sim_{F \cup G} z \land c \quad (\text{if } \bigcap y \sim_{H} z H \not\subseteq F \cup G) \]

Propagation Rules
Properties

Lemma

*The basic constraint system terminates and yields a clause that is equivalent to the first one.*
**Lemma**

The basic constraint system terminates and yields a clause that is equivalent to the first one.

**Lemma**

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that
Properties

Lemma

The basic constraint system terminates and yields a clause that is equivalent to the first one.

Lemma

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that:

- $c$ is in normal form;
The basic constraint system terminates and yields a clause that is equivalent to the first one.

Lemma

Let \( c \) be a clause \( c = g_c \land \exists X \cdot l_c \) such that:

- \( c \) is in normal form;
- \( \forall (g_c) \cap X = \emptyset \);
- every literal in \( l_c \) is about \( X \).
Properties

Lemma

The basic constraint system terminates and yields a clause that is equivalent to the first one.

Lemma

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that:

- $c$ is in normal form;
- $\forall(g_c) \cap X = \emptyset$;
- every literal in $l_c$ is about $X$;
- there is no $y[f]x$ with $x \in X$ and $y \notin X$. 
Properties

Lemma

The basic constraint system terminates and yields a clause that is equivalent to the first one.

Lemma

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that:

- $c$ is in normal form;
- $\forall (g_c) \cap X = \emptyset$;
- every literal in $l_c$ is about $X$;
- there is no $y[f]x$ with $x \in X$ and $y \not\in X$.

Then $c$ is equivalent to $g_c$. 
Table of Contents

1. Description of filesystems
   Unix filesystems
   Static description
   Directory update

2. Constraints
   Definitions
   Basic constraints
   Negation

3. Usages
   Decidability of the First-Order Theory
   Automated Specification for Scripts: Proof of Concept
Negation: new players, new rules

aka *La Slide de la Mort*
Negation: new players, new rules

\[ \neg x[f] y \wedge c \implies (x[f] \uparrow \lor \exists z \cdot (x[f] z \wedge y \not\sim \varnothing z)) \wedge c \]

\[ \neg x[f] \uparrow \wedge c \implies \exists z \cdot x[f] z \wedge c \]

**Simple Replacement Rules**
Negation: new players, new rules

\[ \neg x[f]y \land c \Rightarrow (x[f] \uparrow \lor \exists z \cdot (x[f]z \land y \not\in \emptyset z)) \land c \]
\[ \neg x[f] \uparrow \land c \Rightarrow \exists z \cdot x[f]z \land c \]

Simple Replacement Rules

\[ x[F] \land \neg x[G] \land c \Rightarrow x[F] \land x\langle F \setminus G \rangle \land c \]
\[ x[F] \land x \not\in_G y \land c \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \neq_{F \setminus G} y) \land c \]
\[ x \sim_F y \land x \not\in_G y \land c \Rightarrow x \sim_F y \land x \neq_{F \setminus G} y \land c \]

More Replacement Rules
Negation: new players, new rules

\[
x[F] \land \neg x[G] \land c \implies x[F] \land x\langle F \setminus G \rangle \land c
\]
\[
x[F] \land x \not\forall_G y \land c \implies x[F] \land (\neg y[F \cup G] \lor x \neq_{F\setminus G} y) \land c
\]
\[
x \not\sim_F y \land x \not\forall_G y \land c \implies x \not\sim_F y \land x \neq_{F\setminus G} y \land c
\]

More Replacement Rules
Negation: new players, new rules

\[
x[F] \land \neg x[G] \land c \Rightarrow x[F] \land x\langle F \setminus G \rangle \land c
\]
\[
x[F] \land x \not\in_G y \land c \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \neq_{F \setminus G} y) \land c
\]
\[
x \sim_F y \land x \not\in_G y \land c \Rightarrow x \sim_F y \land x \neq_{F \setminus G} y \land c
\]

More Replacement Rules
Negation: new players, new rules

\[
x(F) := \bigvee_{f \in F} \exists z \cdot x[f]z
\]

\[
x[F] \land \neg x[G] \land c \quad \Rightarrow \quad x[F] \land x(F \setminus G) \land c
\]

\[
x[F] \land x \not\subseteq_G y \land c \quad \Rightarrow \quad x[F] \land (\neg y[F \cup G] \lor x \neq F \setminus G y) \land c
\]

\[
x \not\sim_F y \land x \not\subseteq_G y \land c \quad \Rightarrow \quad x \not\sim_F y \land x \neq F \setminus G y \land c
\]

More Replacement Rules
Negation: new players, new rules

\[ x \not\in_{F} y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f] \circ z') \lor \exists z \cdot (x[f] \downarrow \land y[f] \uparrow) \right) \cup \exists z, z' \cdot (x[f] \downarrow \land y[f] \circ z' \land z \not\in z') \]

- \[ x[F] \land \neg x[G] \land c \Rightarrow x[F] \land x(F \setminus G) \land c \]
- \[ x[F] \land x \not\in_{G} y \land c \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \not\in_{F \setminus G} y) \land c \]
- \[ x \sim_{F} y \land x \not\in_{G} y \land c \Rightarrow x \sim_{F} y \land x \not\in_{F \setminus G} y \land c \]

More Replacement Rules
Negation: new players, new rules

\[ x \not\in_F y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f] z') \lor \exists z \cdot (x[f] z \land y[f] \uparrow) \lor \exists z, z' \cdot (x[f] z \land y[f] z' \land z \not\subseteq z') \right) \]

More Replacement Rules

\[
\begin{align*}
x[F] \land \neg x[G] \land c & \Rightarrow x[F] \land x\langle F \setminus G \rangle \land c \\
x[F] \land x \not\in_G y \land c & \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \not\in_{F \setminus G} y) \land c \\
x \not\in_F y \land x \not\in_G y \land c & \Rightarrow x \not\in_F y \land x \not\in_{F \setminus G} y \land c
\end{align*}
\]

Enlargement and Propagation Rules

\[
\begin{align*}
x \not\in_F y \land \neg x[G'] \land c & \Rightarrow x \not\in_F y \land (\neg x[F \cup G] \lor x\langle F \setminus G' \rangle) \land c \quad (F \not\subseteq G) \\
x \not\in_F y \land \neg x[G'] \land c & \Rightarrow x \not\in_F y \land \neg x[G] \land \neg y[G] \land c \quad (F \subseteq G) \\
x \not\in_F y \land x \not\in_G z \land c & \Rightarrow x \not\in_F y \land (x \not\in_{F \cup G} z \lor x \not\in_{F \setminus G} z) \land c \quad (F \not\subseteq G) \\
x \not\in_F y \land x \not\in_G z \land c & \Rightarrow x \not\in_F y \land x \not\in_G z \land y \not\in_G z \land c \quad (F \subseteq G)
\end{align*}
\]
Negation: new players, new rules

\[ x \not \equiv_F y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f]z') \lor \exists z \cdot (x[f]z \land y[f] \uparrow) \right) \]

\begin{align*}
x[F] \land \neg x[G] \land c & \Rightarrow x[F] \land x(F \setminus G) \land c \\
x[F] \land x \not \in_{F} G y \land c & \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \not \equiv_{F \setminus G} y) \land c \\
x \sim_F y \land x \not \in_{F} G y \land c & \Rightarrow x \sim_F y \land x \not \equiv_{F \setminus G} y \land c
\end{align*}

More Replacement Rules
Negation: new players, new rules

\[ x \not\equiv_F y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f] z') \lor \exists z \cdot (x[f] z \land y[f] \uparrow) \lor \exists z, z' \cdot (x[f] z \land y[f] z' \land z \not\in_{\emptyset} z') \right) \]

\[
\begin{align*}
  x[F] \land \neg x[G] \land c & \implies x[F] \land x[F \setminus G] \land c \\
  x[F] \land x \not\in_{G} y \land c & \implies x[F] \land (\neg y[F \cup G] \lor x \not\equiv_{F \setminus G} y) \land c \\
  x \not\equiv_{F} y \land x \not\in_{G} y \land c & \implies x \not\equiv_{F} y \land x \not\equiv_{F \setminus G} y \land c
\end{align*}

More Replacement Rules
Negation: new players, new rules

\[ x \not\equiv_F y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f]z') \lor \exists z \cdot (x[f]z \land y[f] \uparrow) \right. \]
\[ \left. \lor \exists z, z' \cdot (x[f]z \land y[f]z' \land z \not\sim z') \right) \]

\[
\begin{align*}
    x[F] \land \neg x[G] \land c & \Rightarrow x[F] \land x\langle F \setminus G \rangle \land c \\
    x[F] \land x \not\sim_G y \land c & \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \not\equiv_{F \setminus G} y) \land c \\
    x \sim_F y \land x \not\sim_G y \land c & \Rightarrow x \sim_F y \land x \not\equiv_{F \setminus G} y \land c
\end{align*}
\]

More Replacement Rules

\[ x[F] = \text{"}x\text{ has no feature outside } F\text{"} \]
\[ x \not\sim_G y = \text{"}there is a feature outside } G\text{ that differentiates } x \text{ and } y\text{"} \]
Negation: new players, new rules

\[ x \not\in_F y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f] z') \lor \exists z \cdot (x[f] z \land y[f] \uparrow) \right. \]
\[ \lor \exists z, z' \cdot (x[f] z \land y[f] z' \land z \not\in \emptyset z') \]

\[ x[F] \land \neg x[G] \land c \Rightarrow x[F] \land x\langle F \setminus G \rangle \land c \]
\[ x[F] \land x \not\in_G y \land c \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \not\in_{F \setminus G} y) \land c \]
\[ x \sim_{F} y \land x \not\in_G y \land c \Rightarrow x \sim_{F} y \land x \not\in_{F \setminus G} y \land c \]

More Replacement Rules

\[ x[F] = \text{“}x\text{ has no feature outside } F\text{”} \]
\[ x \not\in_G y = \text{“}there \ is \ a \ feature \ outside \ G \ that \ differentiates \ x \ and \ y\text{”} \]

- either it is in \( F \),
- or it is not,
Negation: new players, new rules

\[ x \not\equiv_F y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f] z') \lor \exists z \cdot (x[f] z \land y[f] \uparrow) \lor \exists z, z' \cdot (x[f] z \land y[f] z' \land z \not\sim \emptyset z') \right) \]

\[ x[F] \land \neg x[G] \land c \Rightarrow x[F] \land x\langle F \setminus G \rangle \land c \]
\[ x[F] \land x \not\sim_G y \land c \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \not\equiv_{F \setminus G} y) \land c \]
\[ x \sim_F y \land x \not\sim_G y \land c \Rightarrow x \sim_F y \land x \not\equiv_{F \setminus G} y \land c \]

**More Replacement Rules**

\[ x[F] = \text{“} x \text{ has no feature outside } F \text{”} \]
\[ x \not\sim_G y = \text{“} \text{there is a feature outside } G \text{ that differentiates } x \text{ and } y \text{”} \]

- either it is in \( F \), and we can list all the cases;
- or it is not,
Negation: new players, new rules

\[ x \not\equiv_F y := \bigvee_{f \in F} \left( \exists z' \cdot (x[f] \uparrow \land y[f]z') \lor \exists z \cdot (x[f]z \land y[f] \uparrow) \lor \exists z, z' \cdot (x[f]z \land y[f]z' \land z \not\sim \emptyset z') \right) \]

\[ x[F] \land \neg x[G] \land c \Rightarrow x[F] \land x(F \setminus G) \land c \]
\[ x[F] \land x \not\sim_G y \land c \Rightarrow x[F] \land (\neg y[F \cup G] \lor x \not\equiv_{F \setminus G} y) \land c \]
\[ x \sim_F y \land x \not\sim_G y \land c \Rightarrow x \sim_F y \land x \not\equiv_{F \setminus G} y \land c \]

More Replacement Rules

\[ x[F] = \text{“} x \text{ has no feature outside } F \text{”} \]
\[ x \not\sim_G y = \text{“} there is a feature outside } G \text{ that differentiates } x \text{ and } y \text{”} \]
- either it is in \( F \), and we can list all the cases;
- or it is not, and since \( x[F] \) then \( \neg y[F \cup G] \).
Lemma

The constraint system terminates and yields a clause that is equivalent to the first one.
Properties

Lemma

The constraint system terminates and yields a clause that is equivalent to the first one.

Lemma

Let \( c \) be a clause \( c = g_c \land \exists X \cdot l_c \) such that:

- \( c \) is in normal form;
- \( \forall (g_c) \cap X = \emptyset \);
- every literal in \( l_c \) is about \( X \);
- there is no \( y[f]x \) with \( x \in X \) and \( y \notin X \).

Then \( c \) is equivalent to \( g_c \).
Does that even terminate?

\[
\text{R-NSim-Fence:}
\]

\[
x[F] \land x \not\sim_G y \land c \\
\Rightarrow x[F] \land (\neg y[F \cup G] \lor x \neq_{F \setminus G} y) \land c
\]
Does that even terminate?

\[ R-\text{NSim-Fence} \text{ (for } F = \{ f \} \text{ and } G = \emptyset): \]

\[
x[\{ f \}] \land x \not\sim \emptyset \ y \land c
\]

\[
\Rightarrow \quad x[\{ f \}] \land (\neg y[\{ f \}] \lor x \neq f \ y) \land c
\]
Does that even terminate?

\[ \text{R-NSim-Fence (for } F = \{ f \} \text{ and } G = \emptyset): \]

\[ x[\{ f \}] \land x \not\sim_{\emptyset} y \land c \]

\[ \Rightarrow \quad \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim_{\emptyset} z' \land x[\{ f \}] \]

...
Does that even terminate?

R-NSIM-FENCE (for $F = \{f\}$ and $G = \emptyset$):

$$x\{f\} \land x \not\in \emptyset \land c \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\in \emptyset z' \land x\{f\}$$

\[
\begin{align*}
x_0[\{f\}] & \quad \not\in \emptyset & \quad y_0 \\
f & \quad \vdash \\
x_1[\{f\}] & \\
f & \quad \vdash \\
x_2[\{f\}] & \\
f & \quad \vdash \\
\vdots & \\
f & \quad \vdash \\
x_n[\{f\}] & \\
f & \quad \vdash \\
\vdots &
\end{align*}
\]
Does that even terminate?

\[
\begin{align*}
&x_0[\{f\}] \quad \cdots \quad \not\emptyset \quad \cdots \quad y_0 \\
&f \quad \vdots \\
&x_1[\{f\}] \\
&f \\
&x_2[\{f\}] \\
&f \\
&\vdots \\
&f \\
&x_n[\{f\}] \\
&f \\
&\vdots \\
\end{align*}
\]

R-NSIM-FENCE (for \( F = \{f\} \) and \( G = \emptyset \)):

\[
x[\{f\}] \land x \not\emptyset y \land c \\
\Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\emptyset z' \land x[\{f\}]
\]

\[\blacktriangleright\] R-NSIM-FENCE with \( x_0 \) and \( y_0 \);
Does that even terminate?

\[ \exists y_1, z_1 \cdot \]

R-NSIM-FENCE (for \( F = \{ f \} \) and \( G = \emptyset \)):

\[ x_0[\{ f \}] \]
\[ f \mid f \]
\[ x_1[\{ f \}] \]
\[ f \mid f \]
\[ x_2[\{ f \}] \]
\[ f \mid f \]
\[ \vdots \]
\[ f \mid f \]
\[ x_n[\{ f \}] \]
\[ f \mid f \]
\[ \vdots \]

\[ x[\{ f \}] \land x \not\sim G \ y \land c \]
\[ \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim G z' \land x[\{ f \}] \]

\[ \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim G z' \land x[\{ f \}] \]

\[ \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim G z' \land x[\{ f \}] \]

\[ \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim G z' \land x[\{ f \}] \]

\[ \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim G z' \land x[\{ f \}] \]

\[ \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim G z' \land x[\{ f \}] \]
Does that even terminate?

\[ \exists y_1, z_1 : \]

\[ x_0[f] \]
\[ f \] \[ f \]
\[ x_1[f] \]
\[ f \]
\[ x_2[f] \]
\[ f \]
\[ \vdots \]
\[ x_n[f] \]
\[ f \]
\[ \vdots \]

R-NSim-Fence (for \( F = \{ f \} \) and \( G = \emptyset \)):

\[ x[f] \land x \not\sim y \land c \]
\[ \Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim z' \land x[f] \]

\[ \blacktriangleright \text{ R-NSim-Fence with } x_0 \text{ and } y_0 ; \]
\[ \blacktriangleright \text{ S-Feats with } x_1 \text{ and } z_1 \]
Does that even terminate?

\[ \exists y_1 \cdot \ldots \]

\[
\begin{array}{c|c|c|c}
 x_0[\{f\}] & f & x_1[\{f\}] & \not\in \emptyset \end{array}
\]

\[
\begin{array}{c|c|c|c|c}
 f & f & y_0 & y_1 \\
 \end{array}
\]

**R-NSim-Fence** (for \( F = \{f\} \) and \( G = \emptyset \)):

\[
x[\{f\}] \land x \not\in \emptyset \land y \land c
\]

\[
\Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\in \emptyset \land z' \land x[\{f\}]
\]

- **R-NSim-Fence** with \( x_0 \) and \( y_0 \);
- **S-Feats** with \( x_1 \) and \( z_1 \);
Does that even terminate?

\[ \exists y_1 \cdot R-\text{NSIM-FENCE (for } F = \{ f \} \text{ and } G = \emptyset): \]

\[
\begin{align*}
& x_0[\{ f \}] & f & y_0 \\
& f & f \\
& x_1[\{ f \}] & \not\emptyset & x_1 \\
& f & f \\
& x_2[\{ f \}] & \not\emptyset & x_2 \\
& f & f \\
& \vdots \\
& x_n[\{ f \}] & f & \vdots \\
\end{align*}
\]

\[
\begin{align*}
& x[\{ f \}] & x \not\emptyset & y \land c \\
\Rightarrow & \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\emptyset z' \land x[\{ f \}] \\
\end{align*}
\]

\[
\begin{align*}
& \blacktriangleright R-\text{NSIM-FENCE with } x_0 \text{ and } y_0; \\
& \blacktriangleright S-\text{FEATS with } x_1 \text{ and } z_1 \\
& \blacktriangleright R-\text{NSIM-FENCE with } x_1 \text{ and } y_1; \\
\end{align*}
\]
Does that even terminate?

\[ \exists y_1, y_2, z_2 : \]

\[ x_0[\{f\}] \]

\[ f \]

\[ x_1[\{f\}] \]

\[ f \]

\[ f \]

\[ x_2[\{f\}] \]

\[ f \]

\[ f \]

\[ z_2 \]

\[ \not\sim \emptyset \]

\[ y_2 \]

\[ \vdots \]

\[ f \]

\[ f \]

\[ f \]

\[ x_n[\{f\}] \]

\[ f \]

\[ f \]

\[ \vdots \]

R-NSIM-FENCE (for \( F = \{f\} \) and \( G = \emptyset \)):

\[ x[\{f\}] \land x \not\sim \emptyset y \land c \]

\[ \Rightarrow \exists z, z' : x[f]z \land y[f]z' \land z \not\sim \emptyset z' \land x[\{f\}] \]

- R-NSIM-FENCE with \( x_0 \) and \( y_0 \);
- S-Feats with \( x_1 \) and \( z_1 \);
- R-NSIM-FENCE with \( x_1 \) and \( y_1 \);
Does that even terminate?

\[ \exists y_1, y_2, z_2 : \]

\[
\begin{array}{c|c}
  x_0[\{f\}] & f \\
  \downarrow f & y_0 \\
  f & f \\
  x_1[\{f\}] & y_1 \\
  \downarrow f & \uparrow f \\
  f & \downarrow f \\
  x_2[\{f\}] & z_2 \not\sim \emptyset y_2 \\
  \downarrow f & \downarrow f \\
  \vdots \\
  \downarrow f & \downarrow f \\
  x_n[\{f\}] & \vdots \\
  \downarrow f & \downarrow f \\
  \vdots \\
\end{array}
\]

\[ \text{R-NSIM-FENCE (for } F = \{f\} \text{ and } G = \emptyset) : \]

\[
x[\{f\}] \land x \not\sim \emptyset y \land c \\
\Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim \emptyset z' \land x[\{f\}] \\
\]

\[ \blacktriangleright \text{ R-NSIM-FENCE with } x_0 \text{ and } y_0 ; \]
\[ \blacktriangleright \text{ S-Feats with } x_1 \text{ and } z_1 \]
\[ \blacktriangleright \text{ R-NSIM-FENCE with } x_1 \text{ and } y_1 ; \]
\[ \blacktriangleright \text{ S-Feats with } x_2 \text{ and } z_2 \]
Does that even terminate?

\[ \exists y_1, y_2 \cdot \forall x_0, x_1, x_2, \ldots, x_n, y_0, y_1, y_2, \ldots \]

\[ R\text{-NSIM-FENCE (for } F = \{f\} \text{ and } G = \emptyset): \]

\[ x_0[\{f\}] \quad f \quad y_0 \]

\[ x_1[\{f\}] \quad f \quad y_1 \]

\[ x_2[\{f\}] \quad \not\sim \quad y_2 \]

\[ \vdots \]

\[ x_n[\{f\}] \quad f \quad \vdots \]

\[ x[\{f\}] \land x \not\sim y \land c \]

\[ \Rightarrow \exists z, z' \cdot x[f] z \land y[f] z' \land z \not\sim z' \land x[\{f\}] \]

\[ \begin{align*}
\blacktriangleright & \quad R\text{-NSIM-FENCE with } x_0 \text{ and } y_0; \\
\blacktriangleright & \quad S\text{-FEATS with } x_1 \text{ and } z_1 \\
\blacktriangleright & \quad R\text{-NSIM-FENCE with } x_1 \text{ and } y_1; \\
\blacktriangleright & \quad S\text{-FEATS with } x_2 \text{ and } z_2
\end{align*} \]
Does that even terminate?

\[ \exists y_1, y_2 \cdot \]

\[
\begin{array}{c|c}
& f \\
\hline
x_0[\{f\}] & y_0 \\
x_1[\{f\}] & y_1 \\
x_2[\{f\}] & \varnothing \\
x_n[\{f\}] & \vdots
\end{array} \]

\( \Rightarrow \quad \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\varnothing z' \land x[\{f\}] \)

\( \quad \Rightarrow \quad \exists \text{R-NSim-Fence (for } F = \{f\} \text{ and } G = \emptyset): \)

\( x[\{f\}] \land x \not\varnothing y \land c \)

\( \Downarrow \quad \text{R-NSim-Fence with } x_0 \text{ and } y_0; \)

\( \Downarrow \quad \text{S-Feats with } x_1 \text{ and } z_1 \)

\( \Downarrow \quad \text{R-NSim-Fence with } x_1 \text{ and } y_1; \)

\( \Downarrow \quad \text{S-Feats with } x_2 \text{ and } z_2 \)

\( \Downarrow \quad \ldots \)
Does that even terminate?

$$\exists y_1, y_2, \ldots, y_n \cdot x_0[f] \land x_{\{f\}} \land y_0 \land c$$

R-NSim-Fence (for $F = \{f\}$ and $G = \emptyset$):

$$x[\{f\}] \land x \not\sim \emptyset y \land c$$

$$\Rightarrow \exists z, z' \cdot x[f]z \land y[f]z' \land z \not\sim \emptyset z' \land x[f]$$

- **R-NSim-Fence** with $x_0$ and $y_0$;
- **S-Feats** with $x_1$ and $z_1$;
- **R-NSim-Fence** with $x_1$ and $y_1$;
- **S-Feats** with $x_2$ and $z_2$;
- **...**
1. Description of filesystems
   Unix filesystems
   Static description
   Directory update

2. Constraints
   Definitions
   Basic constraints
   Negation

3. Usages
   Decidability of the First-Order Theory
   Automated Specification for Scripts: Proof of Concept
Weak Quantifier Elimination

Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$. 

If there is another bloc before, use the given technique:

$\forall \exists \cdots \forall Y \cdot \exists X \cdot c = \Rightarrow \forall \exists \cdots \forall Y, X' \cdot c'$

If not, then it is only a satisfiability question.
Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula
Weak Quantifier Elimination

Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc
Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc:

- Universal

$$\forall \exists \cdots \forall X \cdot c$$
Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc:

- Universal, switch it to existential:

$$\forall \exists \cdots \forall X \cdot c \quad \Longrightarrow \quad \neg \exists \forall \cdots \exists X \cdot \neg c$$
Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc:

- Universal, switch it to existential:

$$\forall \exists \cdots \forall X \cdot c \implies \neg \exists \forall \cdots \exists X \cdot \neg c$$

- Existential:
Weak Quantifier Elimination

Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc:

- Universal, switch it to existential:
  \[
  \forall \exists \cdots \forall X \cdot c \quad \Rightarrow \quad \neg \forall \exists \cdots \exists X \cdot \neg c
  \]

- Existential:
  - If there is an other bloc before
    \[
    \forall \exists \cdots \forall Y \cdot \exists X \cdot c
    \]
Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc:

- Universal, switch it to existential:
  \[
  \forall \exists \cdots \forall X \cdot c \quad \implies \quad \neg \exists \forall \cdots \exists X \cdot \neg c
  \]

- Existential:
  - If there is an other bloc before, use the given technique:
    \[
    \forall \exists \cdots \forall Y \cdot \exists X \cdot c \quad \implies \quad \forall \exists \cdots \forall Y, X' \cdot c'
    \]
Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc:

- Universal, switch it to existential:

  \[
  \forall \exists \cdots \forall X \cdot c \implies \neg \exists \forall \cdots \exists X \cdot \neg c
  \]

- Existential:
  - If there is an other bloc before, use the given technique:

    \[
    \forall \exists \cdots \forall Y \cdot \exists X \cdot c \implies \forall \exists \cdots \forall Y, X' \cdot c'
    \]

  - If not
Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.

Take any closed formula, look at the last quantifier bloc:

- **Universal**, switch it to existential:
  
  \[
  \forall \exists \cdots \forall X \cdot c \implies \neg \exists \cdots \exists X \cdot \neg c
  \]

- **Existential**:
  
  - If there is an other bloc before, use the given technique:
    
    \[
    \forall \exists \cdots \forall Y \cdot \exists X \cdot c \implies \forall \exists \cdots \forall Y, X' \cdot c'
    \]
  
  - If not, then it is only a satisfiability question.
Previous slide said: “Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.”
Previous slide said: “Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.”

Here is what we have:

**Lemma**

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that:

- ... 
- there is no $y[f]x$ with $x \in X$ and $y \notin X$.

Then $c$ is equivalent to $g_c$. 
Previous slide said: “Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.”

Here is what we have:

**Lemma**

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that:

- $\ldots$
- there is no $y[f]x$ with $x \in X$ and $y \notin X$.

Then $c$ is equivalent to $g_c$. 
Previous slide said: “Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.”

Here is what we have:

**Lemma**

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that:

- ... 
- there is no $y[f]x$ with $x \in X$ and $y \notin X$.

Then $c$ is equivalent to $g_c$.

Lukily:

$$\exists X, x \cdot (y[f]x \land c)$$
Previous slide said: “Assume given a technique to transform $\exists X \cdot c$ into an equivalent $\forall X' \cdot c'$.”

Here is what we have:

**Lemma**

Let $c$ be a clause $c = g_c \land \exists X \cdot l_c$ such that:

- ... 
- there is no $y[f]x$ with $x \in X$ and $y \notin X$.

Then $c$ is equivalent to $g_c$.

Lukily:

$$\exists X, x \cdot (y[f]x \land c) \implies \neg y[f] \uparrow \land \forall x \cdot (y[f]x \rightarrow \exists X \cdot c)$$
# Table of Contents

1. Description of filesystems
   - Unix filesystems
   - Static description
   - Directory update

2. Constraints
   - Definitions
   - Basic constraints
   - Negation

3. Usages
   - Decidability of the First-Order Theory
   - Automated Specification for Scripts: Proof of Concept
Demo!