

Unix filesystem and graph constraints

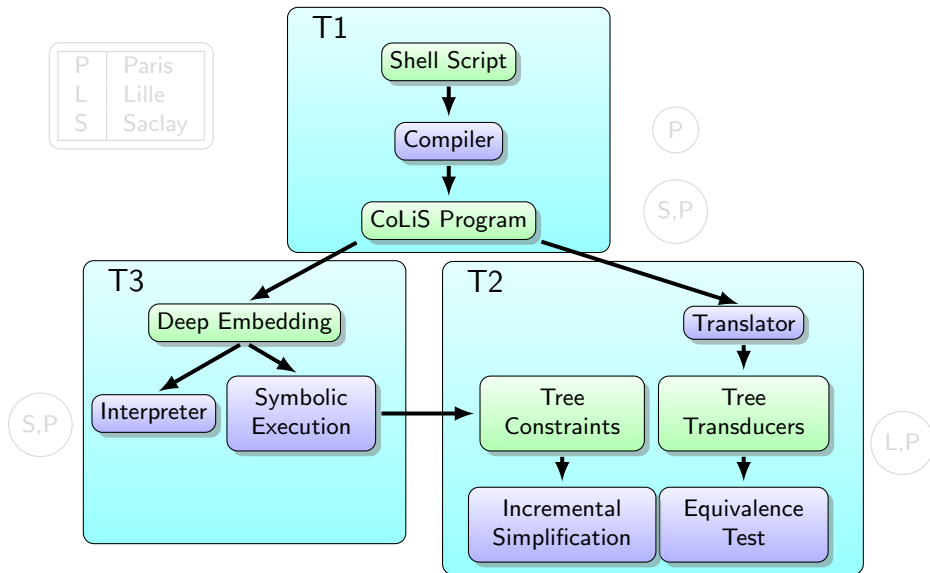
Nicolas Jeannerod



Journées PPS, October 12, 2017

The CoLiS project

P	Paris
L	Lille
S	Saclay



The CoLiS project

P	Paris
L	Lille
S	Saclay

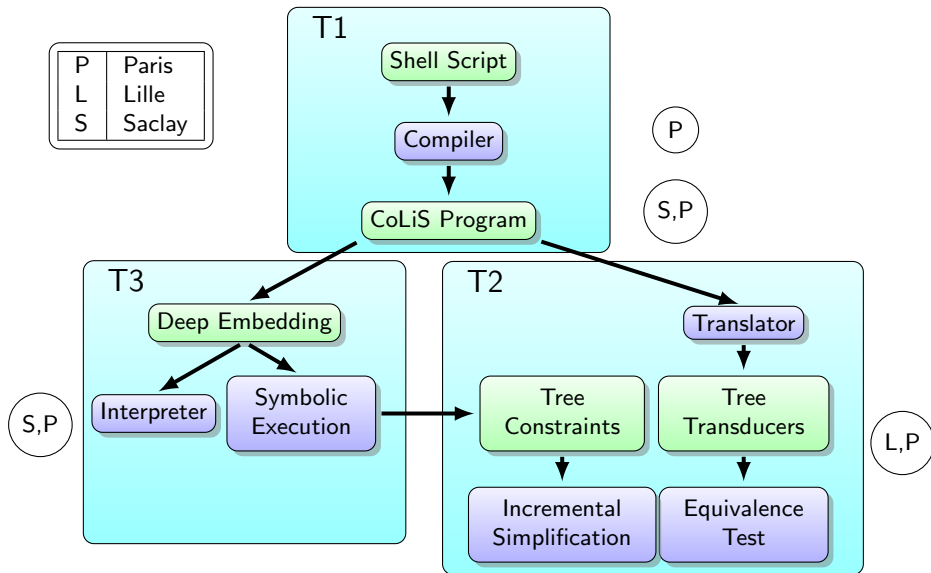
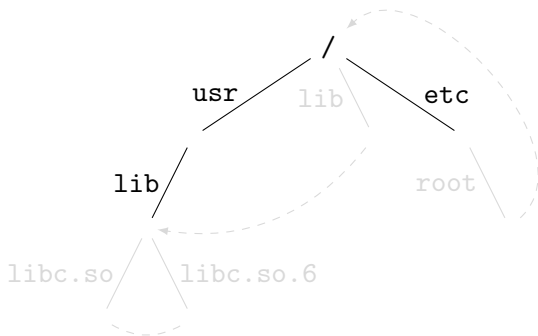


Table of Contents

1. Description of file systems
 - Unix file system
 - Static description
 - Directory update

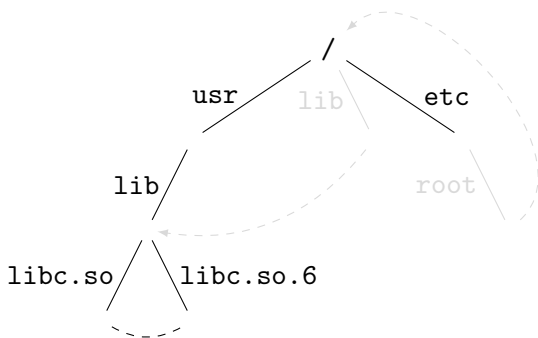
2. Tree constraints
 - Definitions
 - Basic constraints
 - Existential and first order constraints

Unix file system



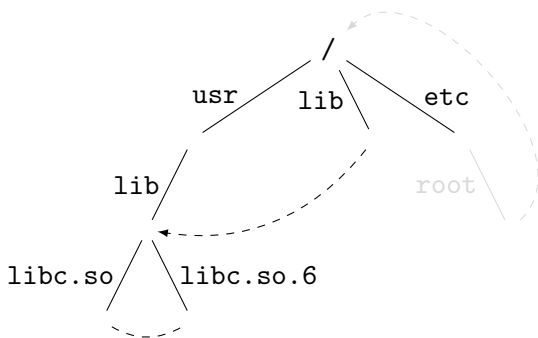
- Basically a tree with labelled nodes and edges;
- There can be sharing at the leafs (hard link between files);
- There can be pointers to other parts of the tree (symbolic links) which may form cycles.

Unix file system



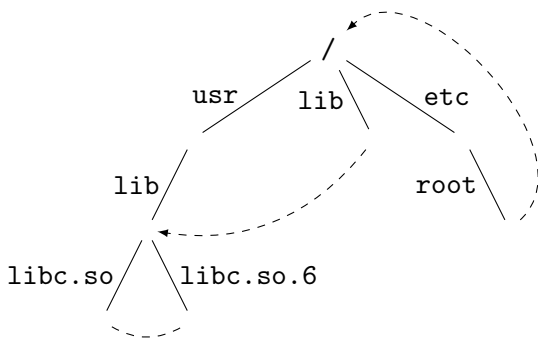
- Basically a tree with labelled nodes and edges;
- There can be sharing at the leafs (hard link between files);
- There can be pointers to other parts of the tree (symbolic links) which may form cycles.

Unix file system



- 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.

Unix file system



- 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

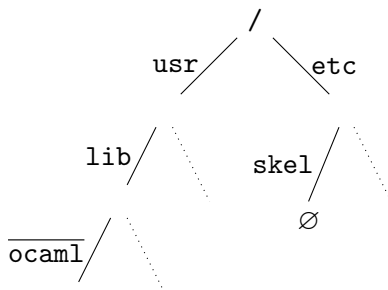
1. Description of file systems

- Unix file system
- **Static description**
- Directory update

2. Tree constraints

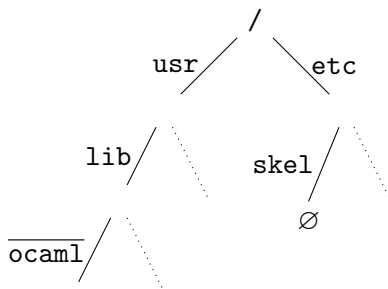
- Definitions
- Basic constraints
- Existential and first order constraints

Static description



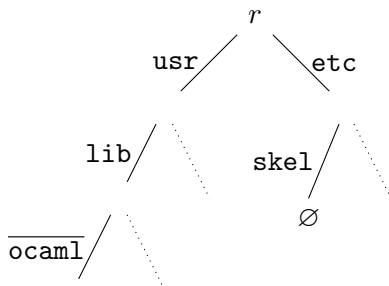
$$r \mid \exists u, v, x, w. \quad r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow \\ \wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

Static description



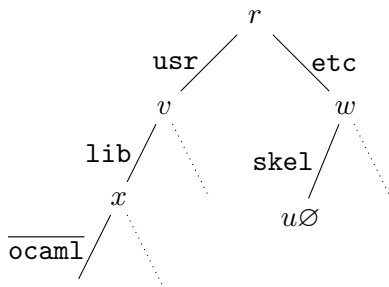
$$r \left| \begin{array}{l} \exists u, v, x, w. \\ r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow \\ \wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u\emptyset \end{array} \right.$$

Static description



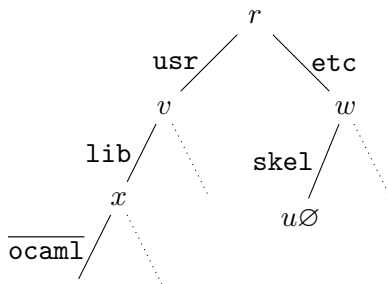
$$r \left| \begin{array}{l} \exists u, v, x, w. \\ r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow \\ \wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u\emptyset \end{array} \right.$$

Static description



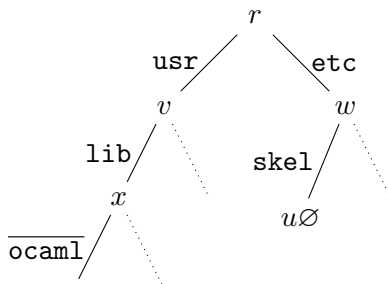
$$r \left| \begin{array}{l} \exists u, v, x, w. \\ r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow \\ \wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u\emptyset \end{array} \right.$$

Static description



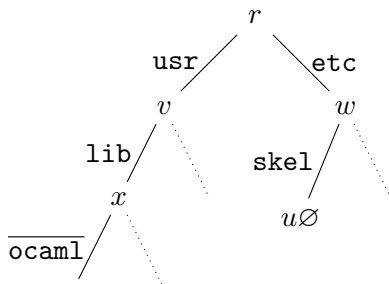
$$r \left| \begin{array}{l} \exists u, v, x, w. \\ r[\mathbf{usr}]v \wedge v[\mathbf{lib}]x \wedge x[\mathbf{ocaml}] \uparrow \\ \wedge r[\mathbf{etc}]w \wedge w[\mathbf{skel}]u \wedge u\emptyset \end{array} \right.$$

Static description



$$r \left| \begin{array}{l} \exists u, v, x, w. \\ r[\mathbf{usr}]v \wedge v[\mathbf{lib}]x \wedge x[\mathbf{ocaml}] \uparrow \\ \wedge r[\mathbf{etc}]w \wedge w[\mathbf{skel}]u \wedge u\emptyset \end{array} \right.$$

Static description



$$r \left| \begin{array}{l} \exists u, v, x, w. \\ r[\mathbf{usr}]v \wedge v[\mathbf{lib}]x \wedge x[\mathbf{ocaml}] \uparrow \\ \wedge r[\mathbf{etc}]w \wedge w[\mathbf{skel}]u \wedge u\emptyset \end{array} \right.$$

Table of Contents

1. Description of file systems

- Unix file system
- Static description
- Directory update

2. Tree constraints

- Definitions
- Basic constraints
- Existential and first order constraints

Directory update



We want something like:

$$r' = r[usr \rightarrow v'] \wedge v' = v[lib \rightarrow x'] \wedge x' = x[ocaml \rightarrow y'] \wedge y' \emptyset$$

Directory update



We want something like:

$$r' = r[usr \rightarrow v'] \wedge v' = v[lib \rightarrow x'] \wedge x' = x[ocaml \rightarrow y'] \wedge y' \emptyset$$

Directory update



We want something like:

$$r' = r[usr \rightarrow v'] \wedge v' = v[lib \rightarrow x'] \wedge x' = x[ocaml \rightarrow y'] \wedge y'\emptyset$$

Directory update



We want something like:

$$r' = r[\mathbf{usr} \rightarrow v'] \wedge v' = v[\mathbf{lib} \rightarrow x'] \wedge x' = x[\mathbf{ocaml} \rightarrow y'] \wedge y' \emptyset$$

Er.. is that really what we want?

- Asymmetric:

$$y = x[f \rightarrow v]$$

- Makes it hard to eliminate variables:

$$y = x[f \rightarrow v] \wedge z = x[g \rightarrow w]$$

- Contains in fact two pieces of information:

- “ y and x are different in f , identical everywhere else”

- “ y points to v through f ”

Er.. is that really what we want?

- Asymmetric:

$$y = x[f \rightarrow v]$$

- Makes it hard to eliminate variables:

$$y = x[f \rightarrow v] \wedge z = x[g \rightarrow w]$$

- Contains in fact two pieces of information:
 - " y and x are different in f , identical everywhere else"
 - " y points to v through f "

Er.. is that really what we want?

- Asymmetric:

$$y = x[f \rightarrow v]$$

- Makes it hard to eliminate variables:

$$y = x[f \rightarrow v] \wedge z = x[g \rightarrow w]$$

- Contains in fact two pieces of information:

- “ y and x are different in f , identical everywhere else”:

$$y \sim_f x$$

- “ y points to v through f ”:

$$y[f]v$$

Er.. is that really what we want?

- Asymmetric:

$$y = x[f \rightarrow v]$$

- Makes it hard to eliminate variables:

$$y = x[f \rightarrow v] \wedge z = x[g \rightarrow w]$$

- Contains in fact two pieces of information:
 - “ y and x are different in f , identical everywhere else”:

$$y \sim_f x$$

- “ y points to v through f ”:

$$y \{f\} v$$

Er.. is that really what we want?

- Asymmetric:

$$y = x[f \rightarrow v]$$

- Makes it hard to eliminate variables:

$$y = x[f \rightarrow v] \wedge z = x[g \rightarrow w]$$

- Contains in fact two pieces of information:
 - “ y and x are different in f , identical everywhere else”:

$$y \sim_f x$$

- “ y points to v through f ”:

$$y[f]v$$

Er.. is that really what we want?

- Asymmetric:

$$y = x[f \rightarrow v]$$

- Makes it hard to eliminate variables:

$$y = x[f \rightarrow v] \wedge z = x[g \rightarrow w]$$

- Contains in fact two pieces of information:
 - “ y and x are different in f , identical everywhere else”:

$$y \sim_f x$$

- “ y points to v through f ”:

$$y[f]v$$

Er.. is that really what we want?

- Asymmetric:

$$y = x[f \rightarrow v]$$

- Makes it hard to eliminate variables:

$$y = x[f \rightarrow v] \wedge z = x[g \rightarrow w]$$

- Contains in fact two pieces of information:

- “ y and x are different in f , identical everywhere else”:

$$y \sim_f x$$

- “ y points to v through f ”:

$$y[f]v$$

Much better

- Allows to express the update:

$$y = x[f \rightarrow v] := y \sim_f x \wedge y[f]v$$

- Symmetric:

$$y \sim_f x \iff x \sim_f y$$

- Transitive:

$$y \sim_f x \wedge z \sim_f x \implies y \sim_f z$$

Much better

- Allows to express the update:

$$y = x[f \rightarrow v] := y \sim_f x \wedge y[f]v$$

- Symmetric:

$$y \sim_f x \iff x \sim_f y$$

- Transitive:

$$y \sim_f x \wedge z \sim_f x \implies y \sim_f z$$

Much better

- Allows to express the update:

$$y = x[f \rightarrow v] := y \sim_f x \wedge y[f]v$$

- Symmetric:

$$y \sim_f x \iff x \sim_f y$$

- Transitive:

$$y \sim_f x \wedge z \sim_f x \implies y \sim_f z$$

Table of Contents

1. Description of file systems
 - Unix file system
 - Static description
 - Directory update

2. Tree constraints
 - **Definitions**
 - Basic constraints
 - Existential and first order constraints

Constraints

- \mathcal{K} finite set; $Dir \in \mathcal{K}$; \mathcal{F} infinite set
- Finite trees labelled with \mathcal{K} on nodes and \mathcal{F} on edges
- x, y variables; $K \in \mathcal{K}$, $f \in \mathcal{F}$, $F \subseteq \mathcal{F}$

Equality	$x \doteq y$	$K(x)$	Kind
Feature	xfy	$xf\uparrow$	Absence
Fence	xF	$x \sim_F y$	Similarity

- Composed with $\neg, \wedge, \vee, \exists x, \forall x$
- No quantification on kinds and features
- Wanted: (un)satisfiability of these constraints
- Bonus point for incremental procedures

Constraints

- \mathcal{K} finite set; $Dir \in \mathcal{K}$; \mathcal{F} infinite set
- Finite trees labelled with \mathcal{K} on nodes and \mathcal{F} on edges
- x, y variables; $K \in \mathcal{K}$, $f \in \mathcal{F}$, $F \subseteq \mathcal{F}$

Equality	$x \doteq y$	$K(x)$	Kind
Feature	xfy	$xf\uparrow$	Absence
Fence	xF	$x \sim_F y$	Similarity

- Composed with $\neg, \wedge, \vee, \exists x, \forall x$
- No quantification on kinds and features
- Wanted: (un)satisfiability of these constraints
- Bonus point for incremental procedures

Constraints

- \mathcal{K} finite set; $Dir \in \mathcal{K}$; \mathcal{F} infinite set
- Finite trees labelled with \mathcal{K} on nodes and \mathcal{F} on edges
- x, y variables; $K \in \mathcal{K}$, $f \in \mathcal{F}$, $F \subseteq \mathcal{F}$

Equality	$x \doteq y$	$K(x)$	Kind
Feature	xfy	$xf\uparrow$	Absence
Fence	xF	$x \sim_F y$	Similarity

- Composed with $\neg, \wedge, \vee, \exists x, \forall x$
- No quantification on kinds and features
- Wanted: (un)satisfiability of these constraints
- Bonus point for incremental procedures

Constraints

- \mathcal{K} finite set; $Dir \in \mathcal{K}$; \mathcal{F} infinite set
- Finite trees labelled with \mathcal{K} on nodes and \mathcal{F} on edges
- x, y variables; $K \in \mathcal{K}$, $f \in \mathcal{F}$, $F \subseteq \mathcal{F}$

Equality	$x \doteq y$	$K(x)$	Kind
Feature	xfy	$xf\uparrow$	Absence
Fence	xF	$x \sim_F y$	Similarity

- Composed with $\neg, \wedge, \vee, \exists x, \forall x$
- No quantification on kinds and features
- Wanted: (un)satisfiability of these constraints
- Bonus point for incremental procedures

Constraints

- \mathcal{K} finite set; $Dir \in \mathcal{K}$; \mathcal{F} infinite set
- Finite trees labelled with \mathcal{K} on nodes and \mathcal{F} on edges
- x, y variables; $K \in \mathcal{K}$, $f \in \mathcal{F}$, $F \subseteq \mathcal{F}$

Equality	$x \doteq y$	$K(x)$	Kind
Feature	xfy	$xf\uparrow$	Absence
Fence	xF	$x \sim_F y$	Similarity

- Composed with $\neg, \wedge, \vee, \exists x, \forall x$
- No quantification on kinds and features
- Wanted: (un)satisfiability of these constraints
- Bonus point for incremental procedures

Game plan

1. Write a system of rewriting rules;
2. Prove that the system terminates (help it if needed);
3. Prove that the rules respect equivalences:

Lemma

If ϕ reduces to ψ , then $\models \phi \leftrightarrow \psi$.

4. Prove nice properties on the normal forms:

Lemma

If ϕ is in normal form, then it is either satisfiable or \perp .

Game plan

1. Write a system of rewriting rules;
2. Prove that the system terminates (help it if needed);
3. Prove that the rules respect equivalences:

Lemma

If ϕ reduces to ψ , then $\models \phi \leftrightarrow \psi$.

4. Prove nice properties on the normal forms:

Lemma

If ϕ is in normal form, then it is either satisfiable or \perp .

Game plan

1. Write a system of rewriting rules;
2. Prove that the system terminates (help it if needed);
3. Prove that the rules respect equivalences:

Lemma

If ϕ reduces to ψ , then $\models \phi \leftrightarrow \psi$.

4. Prove nice properties on the normal forms:

Lemma

If ϕ is in normal form, then it is either satisfiable or \perp .

Game plan

1. Write a system of rewriting rules;
2. Prove that the system terminates (help it if needed);
3. Prove that the rules respect equivalences:

Lemma

If ϕ reduces to ψ , then $\models \phi \leftrightarrow \psi$.

4. Prove nice properties on the normal forms:

Lemma

If ϕ is in normal form, then it is either satisfiable or \perp .

Table of Contents

1. Description of file systems
 - Unix file system
 - Static description
 - Directory update

2. Tree constraints
 - Definitions
 - **Basic constraints**
 - Existential and first order constraints

Basic rules

Basic constraints: conjunction of positive atoms.

SIMPL-FEATS

$$\frac{xfy \wedge xfz}{xfy \wedge y \doteq z}$$

C-FEAT-ABS

$$\frac{xfy \wedge xf\uparrow}{\perp}$$

INTRO-FEAT-SIM

$$\frac{x \sim_F y \wedge xfz}{x \sim_F y \wedge xfz \wedge yfz} \quad f \notin F$$

INTRO-SIM-SIMS

$$\frac{x \sim_F y \wedge y \sim_G z}{x \sim_F y \wedge y \sim_G z \wedge x \sim_{(FUG)} z}$$

Basic rules

Basic constraints: conjunction of positive atoms.

SIMPL-FEATS

$$\frac{xfy \wedge xfz}{xfy \wedge y \doteq z}$$

C-FEAT-ABS

$$\frac{xfy \wedge xf\uparrow}{\perp}$$

INTRO-FEAT-SIM

$$\frac{x \sim_F y \wedge xfz}{x \sim_F y \wedge xfz \wedge yfz} \quad f \notin F$$

INTRO-SIM-SIMS

$$\frac{x \sim_F y \wedge y \sim_G z}{x \sim_F y \wedge y \sim_G z \wedge x \sim_{(FUG)} z}$$

Basic rules

Basic constraints: conjunction of positive atoms.

SIMPL-FEATS

$$\frac{xfy \wedge xfz}{xfy \wedge y \doteq z}$$

C-FEAT-ABS

$$\frac{xfy \wedge xf\uparrow}{\perp}$$

INTRO-FEAT-SIM

$$\frac{x \sim_F y \wedge xfz}{x \sim_F y \wedge xfz \wedge yfz} \quad f \notin F$$

INTRO-SIM-SIMS

$$\frac{x \sim_F y \wedge y \sim_G z}{x \sim_F y \wedge y \sim_G z \wedge x \sim_{(FUG)} z}$$

Basic rules

Basic constraints: conjunction of positive atoms.

$$\frac{\text{SIMPL-FEATS} \quad xfy \wedge xfz}{xfy \wedge y \doteq z}$$

$$\frac{\text{C-FEAT-ABS} \quad xfy \wedge xf\uparrow}{\perp}$$

$$\frac{\text{INTRO-FEAT-SIM} \quad x \sim_F y \wedge xfz}{x \sim_F y \wedge xfz \wedge yfz} \quad f \notin F$$

$$\frac{\text{INTRO-SIM-SIMS} \quad x \sim_F y \wedge y \sim_G z}{x \sim_F y \wedge y \sim_G z \wedge x \sim_{(F \cup G)} z}$$

Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow$$

$$\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

$$\wedge \dots$$


Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow$$

$$\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

$$\wedge \dots$$

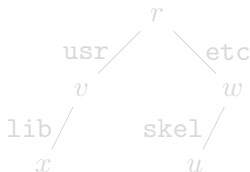

Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow$$

$$\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

$$\wedge \dots$$


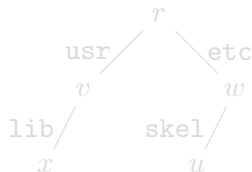
Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow$$

$$\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

$$\wedge \dots$$


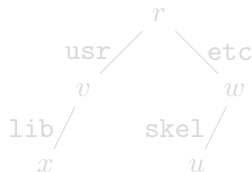
Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow$$

$$\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

$$\wedge \dots$$


Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\mathbf{usr}]v \wedge v[\mathbf{lib}]x \wedge x[\mathbf{ocaml}] \uparrow$$

$$\wedge r[\mathbf{etc}]w \wedge w[\mathbf{skel}]u \wedge u\emptyset$$

$$\wedge \dots$$

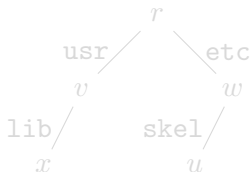

Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow$$

$$\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

$$\wedge \dots$$


Basic constraints

Basic constraints: conjunction of positive atoms

- Equality: rewritten
- Kind: Static “positive” information
- Feature: Static “positive” information
- Absence: Static “negative” information
- Fence: Static “negative” information
- Similarity: **Dynamic** information

$$r[\text{usr}]v \wedge v[\text{lib}]x \wedge x[\text{ocaml}] \uparrow$$

$$\wedge r[\text{etc}]w \wedge w[\text{skel}]u \wedge u \emptyset$$

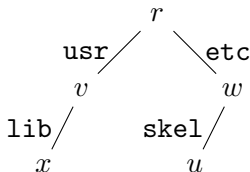
$$\wedge \dots$$


Table of Contents

1. Description of file systems
 - Unix file system
 - Static description
 - Directory update

2. Tree constraints
 - Definitions
 - Basic constraints
 - Existential and first order constraints

Negation: new players, new rules

- $\neg xF$: there exist a $g \notin F$ such that $xg \downarrow$;
- $x \not\sim_F y$: there exist a $g \notin F$ such that $x \not\sim_g y$;

$$\frac{\text{REPL-NKIND} \quad \neg K(x)}{\bigvee_{L \in \mathcal{K}} L(x)}$$

$$\frac{\text{REPL-NABS} \quad \neg x f \uparrow}{\exists z. x f z}$$

$$\frac{\text{REPL-NSIM-NFENCE} \quad xF \wedge x \not\sim_G y}{xF \wedge \neg xG} \quad F \subseteq G$$

Negation: new players, new rules

- $\neg xF$: there exist a $g \notin F$ such that $xg \downarrow$;
- $x \not\sim_F y$: there exist a $g \notin F$ such that $x \not\sim_g y$;

$$\frac{\text{REPL-NKIND} \quad \neg K(x)}{\bigvee_{L \in \mathcal{K}} L(x)}$$

$$\frac{\text{REPL-NABS} \quad \neg x f \uparrow}{\exists z. x f z}$$

$$\frac{\text{REPL-NSIM-NFENCE} \quad xF \wedge x \not\sim_G y}{xF \wedge \neg xG} \quad F \subseteq G$$

Negation: new players, new rules

- $\neg xF$: there exist a $g \notin F$ such that $xg \downarrow$;
- $x \not\sim_F y$: there exist a $g \notin F$ such that $x \not\sim_g y$;

$$\frac{\text{REPL-NKIND} \quad \neg K(x)}{\bigvee_{L \in \mathcal{K}} L(x)}$$

$$\frac{\text{REPL-NABS} \quad \neg x f \uparrow}{\exists z. x f z}$$

$$\frac{\text{REPL-NSIM-NFENCE} \quad xF \wedge x \not\sim_G y}{xF \wedge \neg xG} \quad F \subseteq G$$

Negation: new players, new rules

- $\neg xF$: there exist a $g \notin F$ such that $xg \downarrow$;
- $x \not\sim_F y$: there exist a $g \notin F$ such that $x \not\sim_g y$;

$$\frac{\text{REPL-NKIND} \quad \neg K(x)}{\bigvee_{L \in \mathcal{K}} L(x)}$$

$$\frac{\text{REPL-NABS} \quad \neg x f \uparrow}{\exists z. x f z}$$

$$\frac{\text{REPL-NSIM-NFENCE} \quad xF \wedge x \not\sim_G y}{xF \wedge \neg xG} \quad F \subseteq G$$

Quantifier elimination

- Goal: be able to change an existentially quantified block into a universally quantified one.

$$\left(\exists \vec{X}. \bigwedge \dots\right) \leftrightarrow \left(\forall \vec{Y}. \bigvee \bigwedge \dots\right)$$

- Special rules:

ENLARG-FEAT-LOCAL

$$\frac{\exists x. \exists \vec{X}. (yfx \wedge \phi(x, \vec{X}))}{yf \downarrow \wedge \forall x. \exists \vec{X}. (yfx \rightarrow \phi(x, \vec{X}))}$$

Quantifier elimination

- Goal: be able to change an existentially quantified block into a universally quantified one.

$$\left(\exists \vec{X}. \bigwedge \dots\right) \leftrightarrow \left(\forall \vec{Y}. \bigvee \bigwedge \dots\right)$$

- Special rules:

$$\frac{\text{ENLARG-FEAT-LOCAL} \quad \exists x. \exists \vec{X}. (yfx \wedge \phi(x, \vec{X}))}{yf \downarrow \wedge \forall x. \exists \vec{X}. (yfx \rightarrow \phi(x, \vec{X}))}$$

Lemma of 31 August

Lemma (31 August)

Let ϕ be a conjunction of the form:

$$\phi(\vec{X}, \vec{Y}) = \left(\bigwedge \text{stuff about } \vec{X} \right) \wedge \psi(\vec{Y})$$

in normal form for our system.

Then we have:

$$\models \forall \vec{Y}. \left(\exists \vec{X}. \phi(\vec{X}, \vec{Y}) \right) \leftrightarrow \psi(\vec{Y})$$

- The system propagates all the useful information
- We can just remove what we don't need!

Lemma of 31 August

Lemma (31 August)

Let ϕ be a conjunction of the form:

$$\phi(\vec{X}, \vec{Y}) = \left(\bigwedge \text{stuff about } \vec{X} \right) \wedge \psi(\vec{Y})$$

in normal form for our system.

Then we have:

$$\models \forall \vec{Y}. \left(\exists \vec{X}. \phi(\vec{X}, \vec{Y}) \right) \leftrightarrow \psi(\vec{Y})$$

- The system propagates all the useful information
- We can just remove what we don't need!

Thank you for your attention!

Recap':

- Need constraints on graphs to represent relations on file systems;
- Extend “feature trees” with $x \sim_F y$ (“ x and y are the same, except maybe for the features in F ”);
- Use a system of rewrite rules whose normal forms have nice properties.

Future work:

- Cleanup, formalise in a technical report;
- Add inodes, permissions, timestamps, etc.
- Implement an efficient version for the existential subset.