The Specification of POSIX File Systems

Gian Ntzik, Pedro da Rocha Pinto and Philippa Gardner

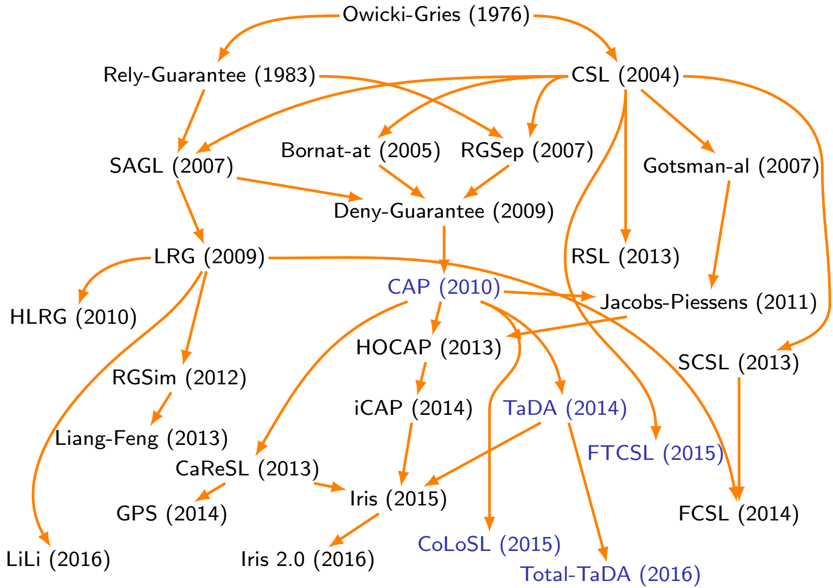
  

Imperial College London

gian.ntzik08@imperial.ac.uk
pedro.da-rocha-pinto09@imperial.ac.uk
p.gardner@imperial.ac.uk

INVEST 2017

Logics for Specifying Concurrent Programs



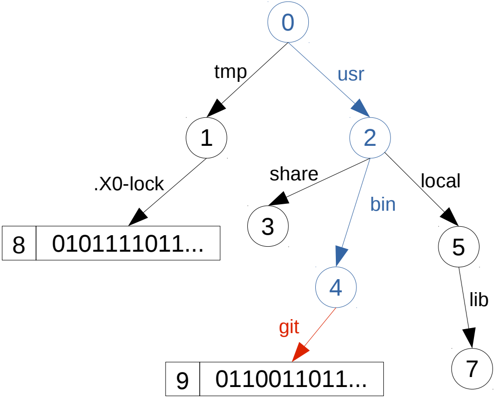
Thanks, Ilya Sergey.

POSIX File Systems

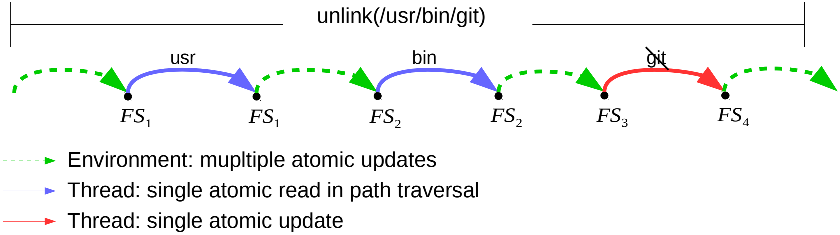
- The POSIX standard is written in English
- Several formal sequential specifications have been given:
  - not good for client reasoning if based on first-order logic;
  - good for client reasoning if based on separation logic.
- The English specification of the concurrent behaviour is poor.
- We have the first concurrent specification of POSIX file systems.

A POSIX operation: unlink(/usr/bin/git)

unlink(/usr/bin/git) takes a sequence of atomic actions



unlink(/usr/bin/git): a Sequence of Atomic Actions



Concurrent Specification of unlink(*path*)

$$
\begin{aligned}
\texttt{unlink}(path) \sqsubseteq\ & \textbf{let } p = \texttt{dirname}(path);\\
& \textbf{let } a = \texttt{basename}(path);\\
& \textbf{let } r = \texttt{resolve}(p, \iota_0);\\
& \textbf{if } \neg \mathsf{iserr}(r) \textbf{ then}\\
& \quad \texttt{return } \texttt{link\_delete}(r, a)\\
& \qquad\qquad \sqcup\, \texttt{link\_delete\_notdir}(r, a)\\
& \textbf{else } \texttt{return } r \textbf{ fi}
\end{aligned}
$$

- $\texttt{link\_delete}(r, a)$ atomically removes link a, even if it links a directory
- $\texttt{link\_delete\_notdir}(r, a)$ atomically removes link a, only if it does not link a directory
- $\sqcup$: non-deterministic *angelic choice*

Concurrent Specification of path resolution

$$
\begin{array}{l}
\textbf{letrec}\ \texttt{resolve}(path, \iota) \triangleq \\
\quad \textbf{if}\ path = \texttt{null}\ \textbf{then}\ \texttt{return}\ \iota\ \textbf{else} \\
\quad\quad \textbf{let}\ a = \texttt{head}(path); \\
\quad\quad \textbf{let}\ p = \texttt{tail}(path); \\
\quad\quad \textbf{let}\ r = \texttt{link\_lookup}(\iota, a); \\
\quad\quad \textbf{if}\ \mathrm{iserr}(r)\ \textbf{then}\ \texttt{return}\ r \\
\quad\quad \textbf{else}\ \texttt{return}\ \texttt{resolve}(p, r)\ \textbf{fi} \\
\quad \textbf{fi}
\end{array}
$$

- $\texttt{link\_lookup}(\iota, a)$ **atomically** lookup the link named a in the directory with inode ι.

Concurrent Specification of $\texttt{unlink}(\mathit{path})$

$$
\begin{aligned}
\texttt{unlink}(\mathit{path}) \sqsubseteq\ & \textbf{let } p = \texttt{dirname}(\mathit{path}); \\
& \textbf{let } a = \texttt{basename}(\mathit{path}); \\
& \textbf{let } r = \texttt{resolve}(p, \iota_0); \\
& \textbf{if } \neg\mathrm{iserr}(r) \textbf{ then} \\
& \quad \texttt{return } \texttt{link\_delete}(r, a) \\
& \qquad\qquad \sqcup \texttt{link\_delete\_notdir}(r, a) \\
& \textbf{else } \texttt{return } r \textbf{ fi}
\end{aligned}
$$

Concurrent Specification of $\texttt{unlink}(\mathit{path})$

$$
\begin{aligned}
\texttt{unlink}(\mathit{path}) \sqsubseteq\ & \textbf{let } p = \texttt{dirname}(\mathit{path});\\
& \textbf{let } a = \texttt{basename}(\mathit{path});\\
& \textbf{let } r = \texttt{resolve}(p, \iota_0);\\
& \textbf{if } \neg \mathrm{iserr}(r) \textbf{ then}\\
& \quad \texttt{return } \texttt{link\_delete}(r, a)\\
& \qquad\qquad\quad \sqcup\, \texttt{link\_delete\_notdir}(r, a)\\
& \textbf{else } \texttt{return } r \textbf{ fi}
\end{aligned}
$$

$\texttt{link\_delete}(\iota, a) \triangleq$

$$
\begin{aligned}
& \mathbb{V} FS.\, \langle \mathrm{fs}(FS) \wedge \mathrm{isdir}(FS(\iota)) \,, a \in FS(\iota) \Longrightarrow \mathrm{fs}(FS[\iota \mapsto FS(\iota) \setminus \{a\}]) * \texttt{ret} = 0 \rangle\\
& \sqcap\, \texttt{return } \texttt{enoent}(\iota, a)\\
& \sqcap\, \texttt{return } \texttt{enotdir}(\iota)
\end{aligned}
$$

$\texttt{enoent}(\iota, a)$ and $\texttt{enotdir}(\iota)$ describe error cases. $\sqcap$ is non-deterministic *demonic* choice.

Concurrent specification of `link`(*source*, *target*)

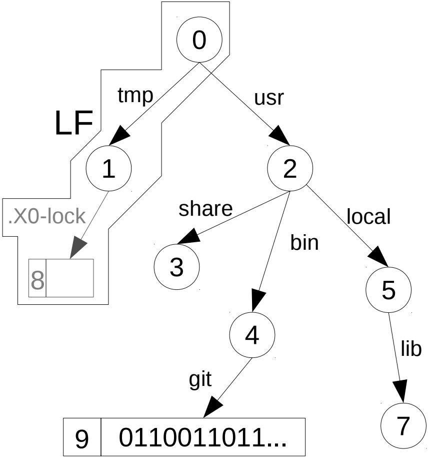
$$
\begin{array}{l}
\texttt{link}(source, target) \\
\quad \sqsubseteq \textbf{let } p_s = \texttt{dirname}(source); \textbf{let } a = \texttt{basename}(source); \\
\qquad \textbf{let } p_t = \texttt{dirname}(target); \textbf{let } b = \texttt{basename}(target); \\
\qquad \textbf{let } r_s, r_t = \texttt{resolve}(p_s, \iota_0) \parallel \texttt{resolve}(p_t, \iota_0); \\
\qquad \textbf{if } \neg\text{iserr}(r_s) \land \neg\text{iserr}(r_t) \textbf{ then} \\
\qquad\quad \texttt{return } \texttt{link\_insert}(r_s, a, r_t, b) \\
\qquad\qquad\qquad \sqcup\ \texttt{link\_insert\_notdir}(r_s, a, r_t, b) \\
\qquad \textbf{else if } \text{iserr}(r_s) \land \neg\text{iserr}(r_t) \textbf{ then } \texttt{return } r_s \\
\qquad \textbf{else if } \neg\text{iserr}(r_s) \land \text{iserr}(r_t) \textbf{ then } \texttt{return } r_t \\
\qquad \textbf{else if } \text{iserr}(r_s) \land \text{iserr}(r_t) \textbf{ then } \texttt{return } r_s \sqcup \texttt{return } r_t \textbf{ fi}
\end{array}
$$

Lock-file Client

- `lock(path)`: atomically create a non-existing lock file at `path`
- `unlock(path)`: remove the lock file identified by `path`
- Implemented similarly to spin locks
  - `open(path, O_CREAT|O_EXCL)` to try to lock
  - `unlink` to unlock

Lock-file Protocol Agreement

We want the module and the environment to agree on a boundary



Lock-file Specification

$$\mathsf{LF}(path) \vdash \texttt{lock}(path) \sqsubseteq \mathbb{V} v \in \{0,1\}. \left\langle \mathsf{Lock}(path, v)\ ,\ \ \mathsf{Lock}(path, 1) * v = 0 \right\rangle$$

$$\mathsf{LF}(path) \vdash \texttt{unlock}(path) \sqsubseteq \left\langle \mathsf{Lock}(path, 1)\ ,\ \ \mathsf{Lock}(path, 0) \right\rangle$$

- LF(*path*) is a *context invariant* denoting what the implementation and the environment can and cannot do to the path:
  - The path-prefix of *path* is not changed (everyone can only read)
  - Only the module create the lock file at *path* (locks the lock)
  - Only the module removes the lock file at *path* (unlocks the lock)

Conclusions

- Concurrent specification of the POSIX file system (fragment)
- Combination of the refinement calculus and a modern concurrent separation logic called TaDA
- Client reasoning: lock files, named pipes, email server
- Future: executable specification which we can test against implementations
- Future: now what about distribution....