Compilation Semantics for a Programming Language with Versions

🎤**Yudai Tanabe**[1], Luthfan Anshar Lubis[2], Tomoyuki Aotani[3], Hidehiko Masuhara[2]

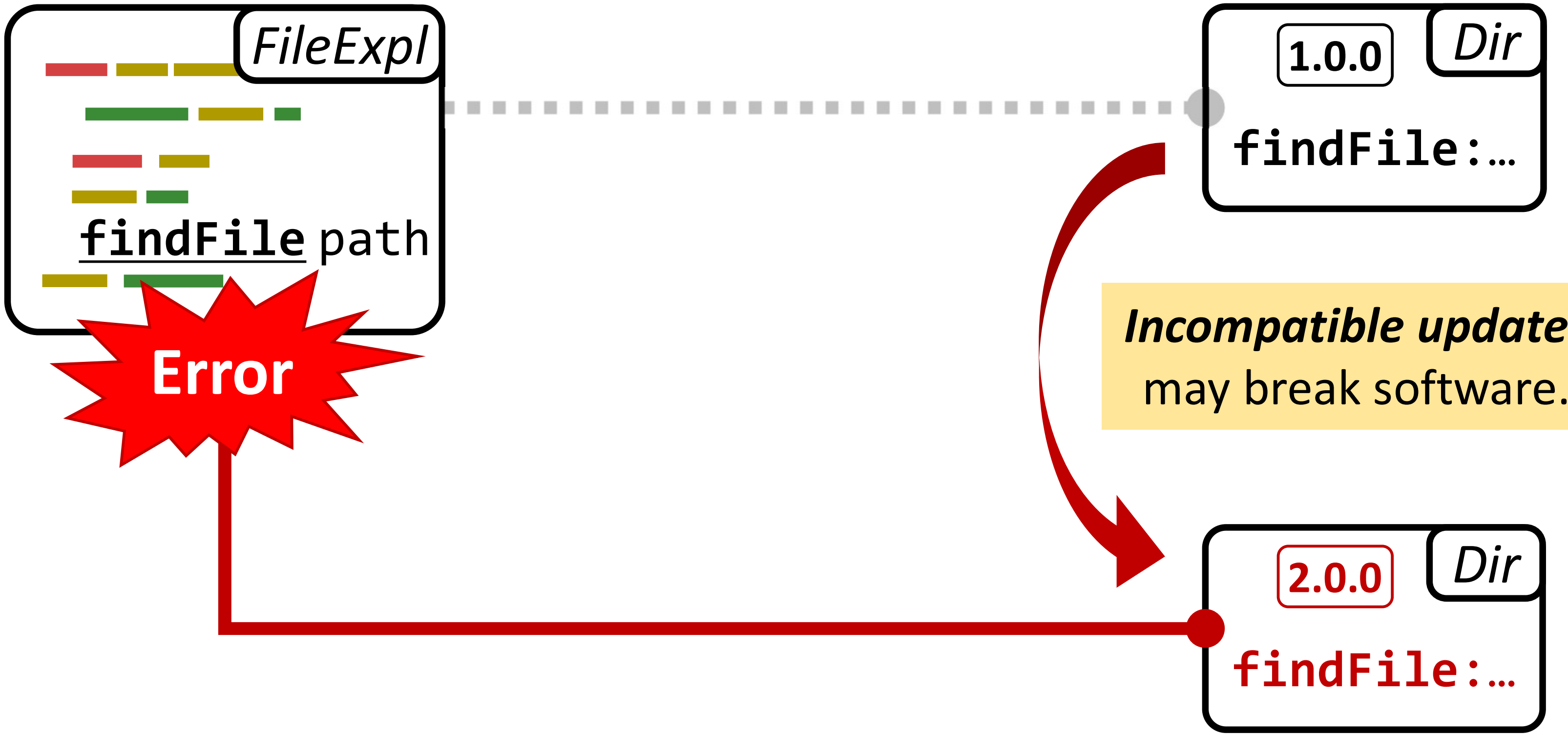
[1]**Kyoto University**, [2]Tokyo Institute of Technology, [3]Sanyo-Onoda City University

APLAS2023 @ Taipei
November 27, 2023

Update Dilemma: Enhancements *vs.* Adaptation Costs

[Werner’13, Bavota’15]

Intricate updating processes are deterring programmers from updates.



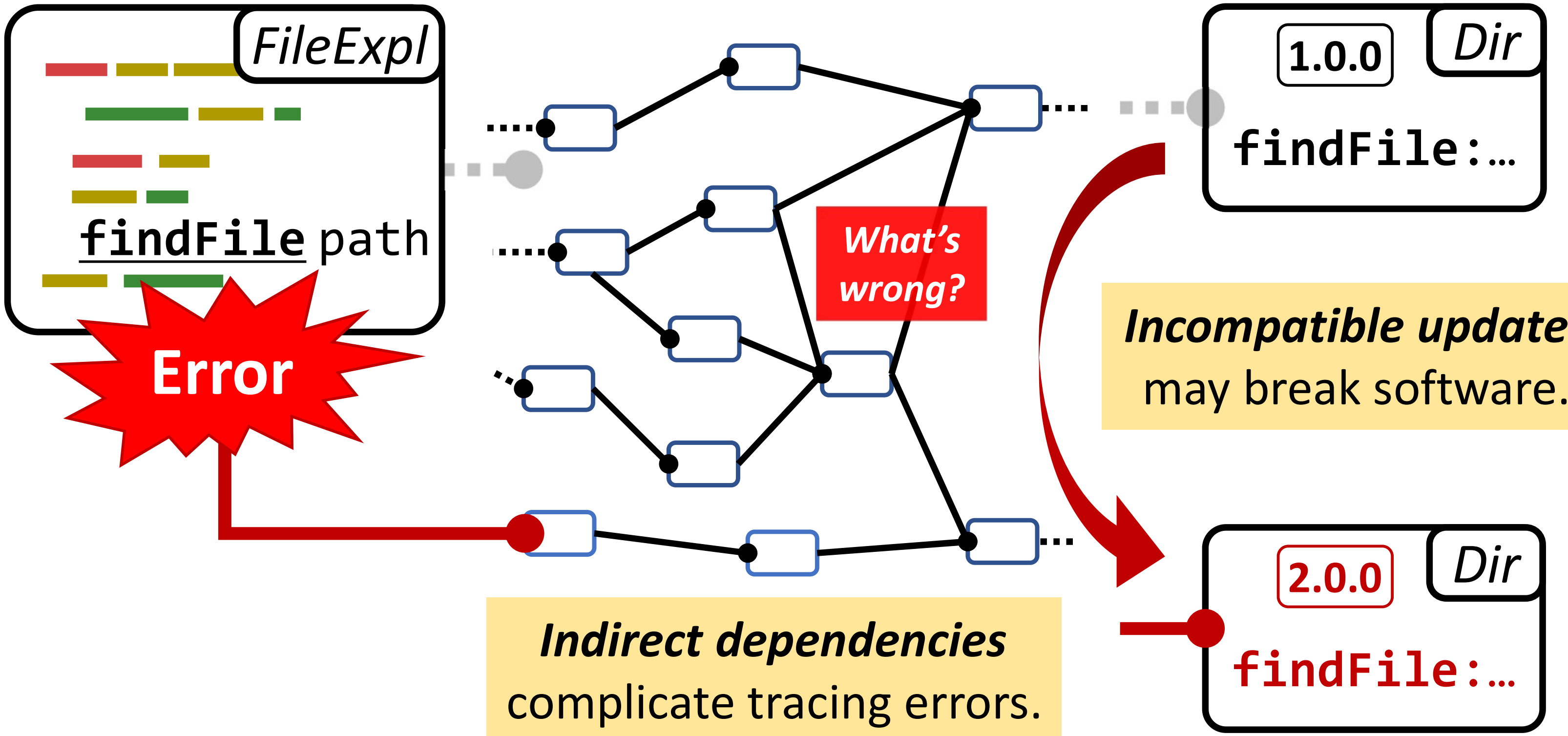
Incompatible updates may break software.

Background

Update Dilemma: Enhancements *vs.* Adaptation Costs

[Werner'13, Bavota'15]

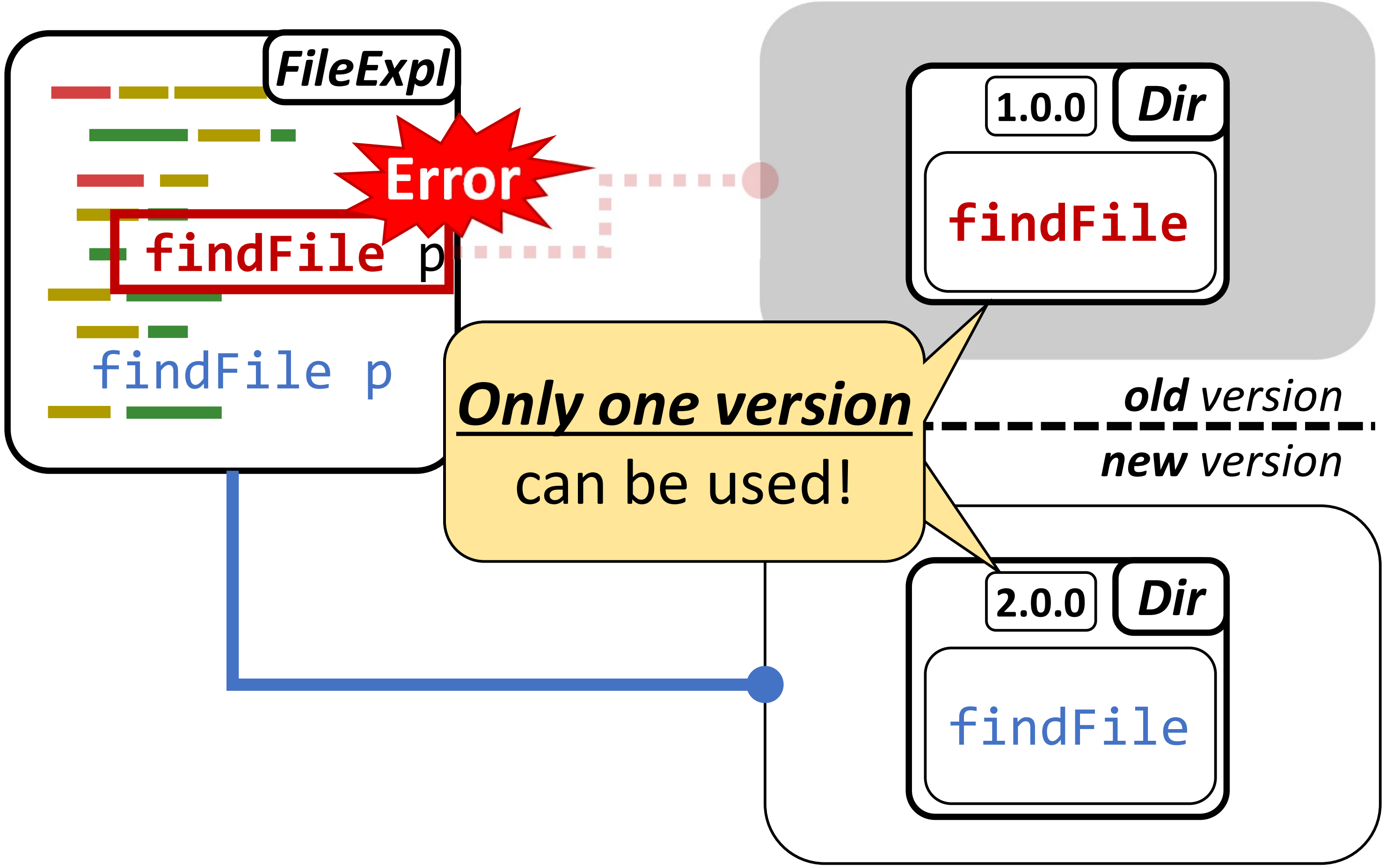
Intricate updating processes are deterring programmers from updates.



Incompatible updates may break software.

Indirect dependencies complicate tracing errors.

One-version-at-a-time Limitation



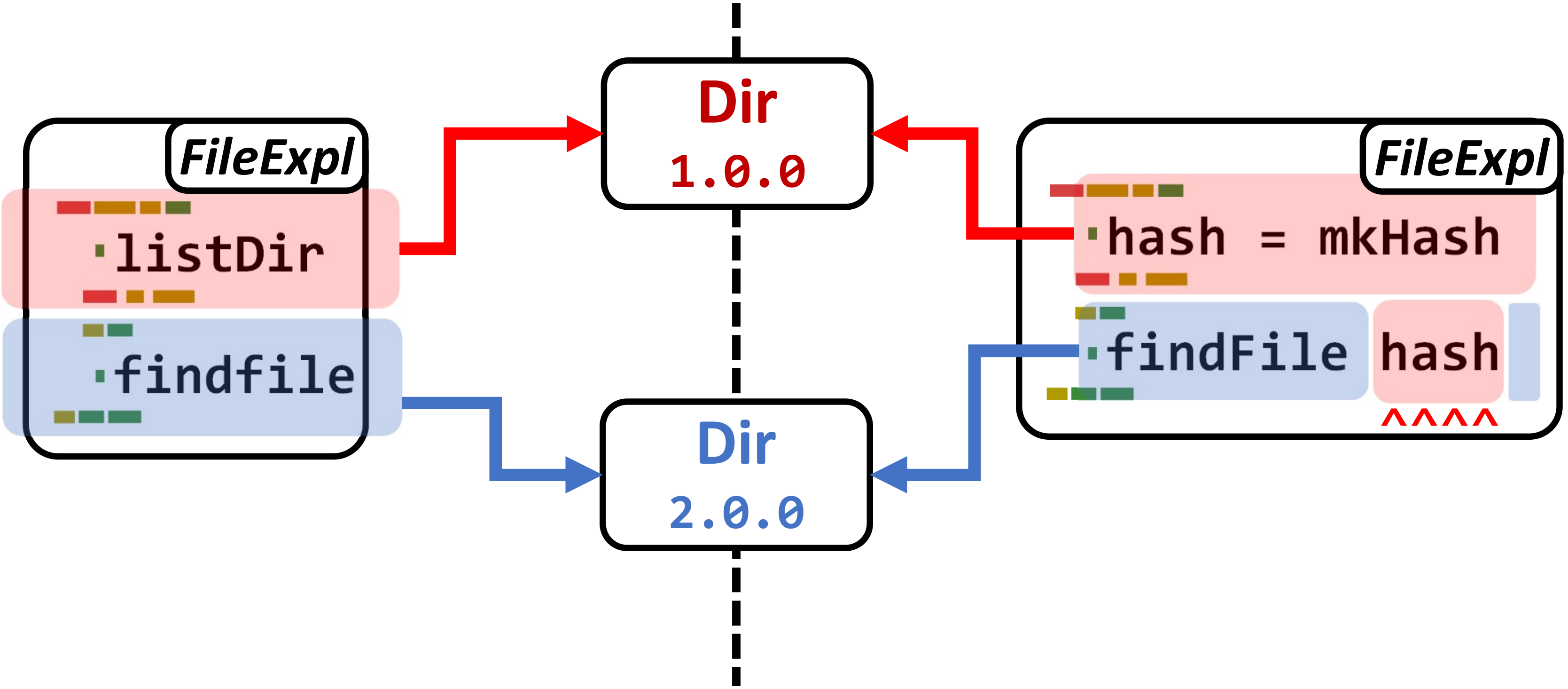
Only one version can be used!

Background – [‹Programming›'22]

Programming with Versions (PwV)

Goal

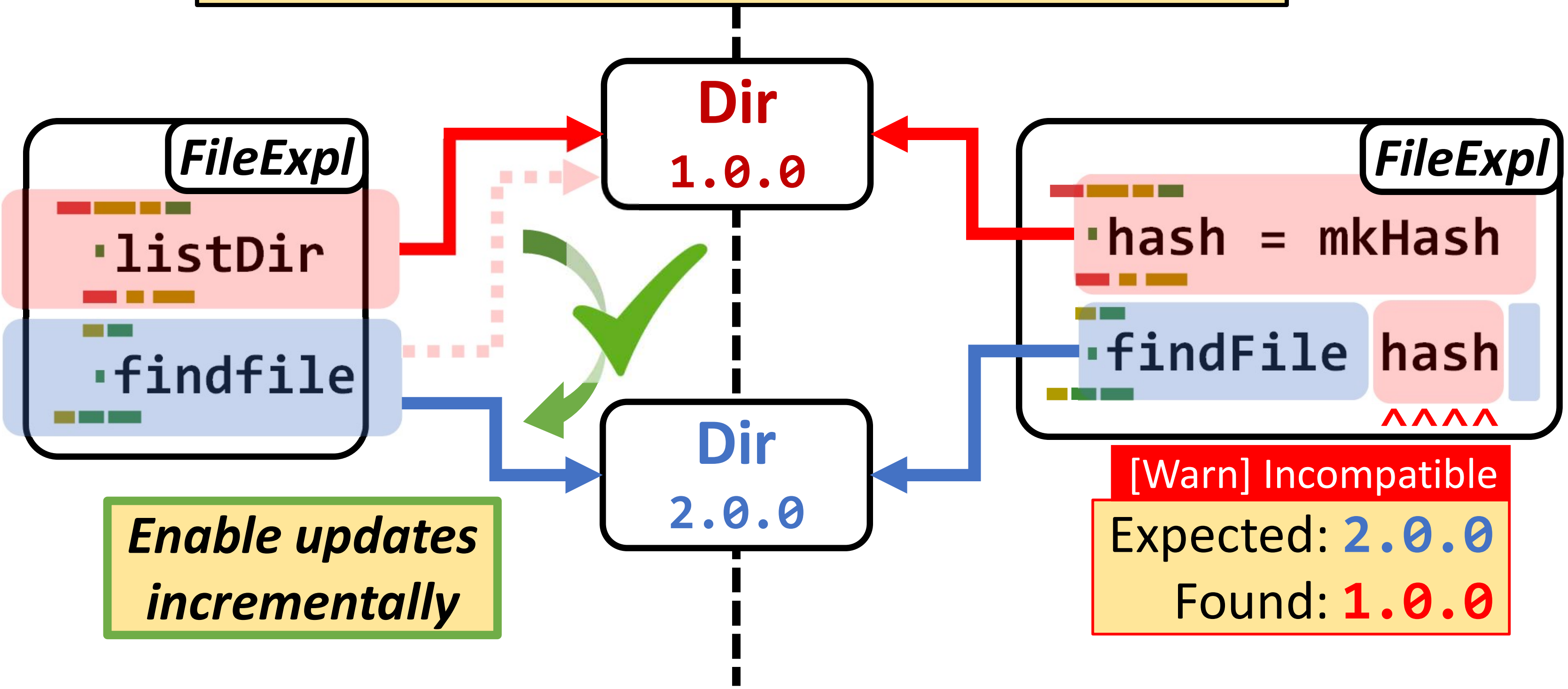
- ***Handling multiple versions*** in one client
- ***Detecting incompatible version*** usage



Background

Why Language-based Approach?

Benefit ***Split huge update into smaller and consistent tasks***



Enable updates incrementally

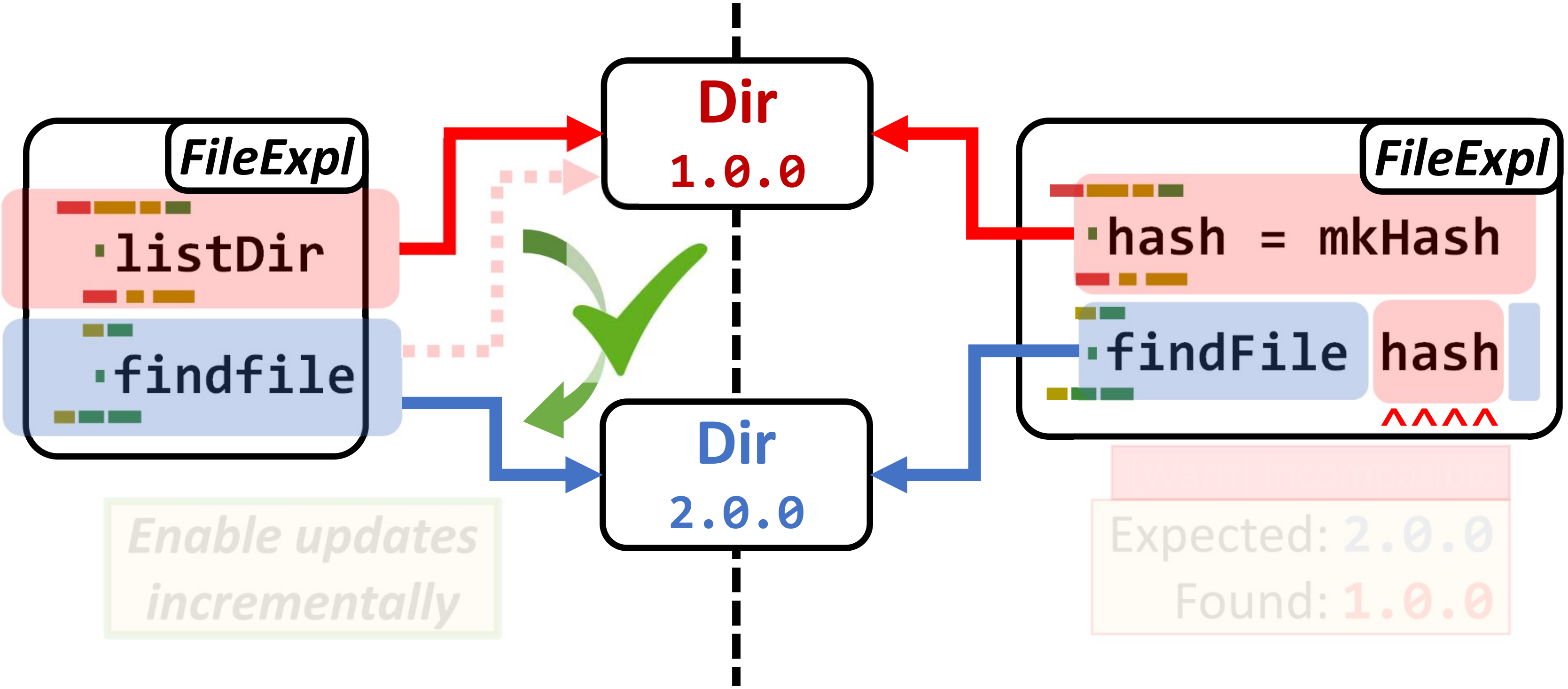
[Warn] Incompatible
Expected: 2.0.0
Found: 1.0.0

Background

Existing PwV Languages

Goal

- *Handling multiple versions* in one client
- *Detecting incompatible version* usage

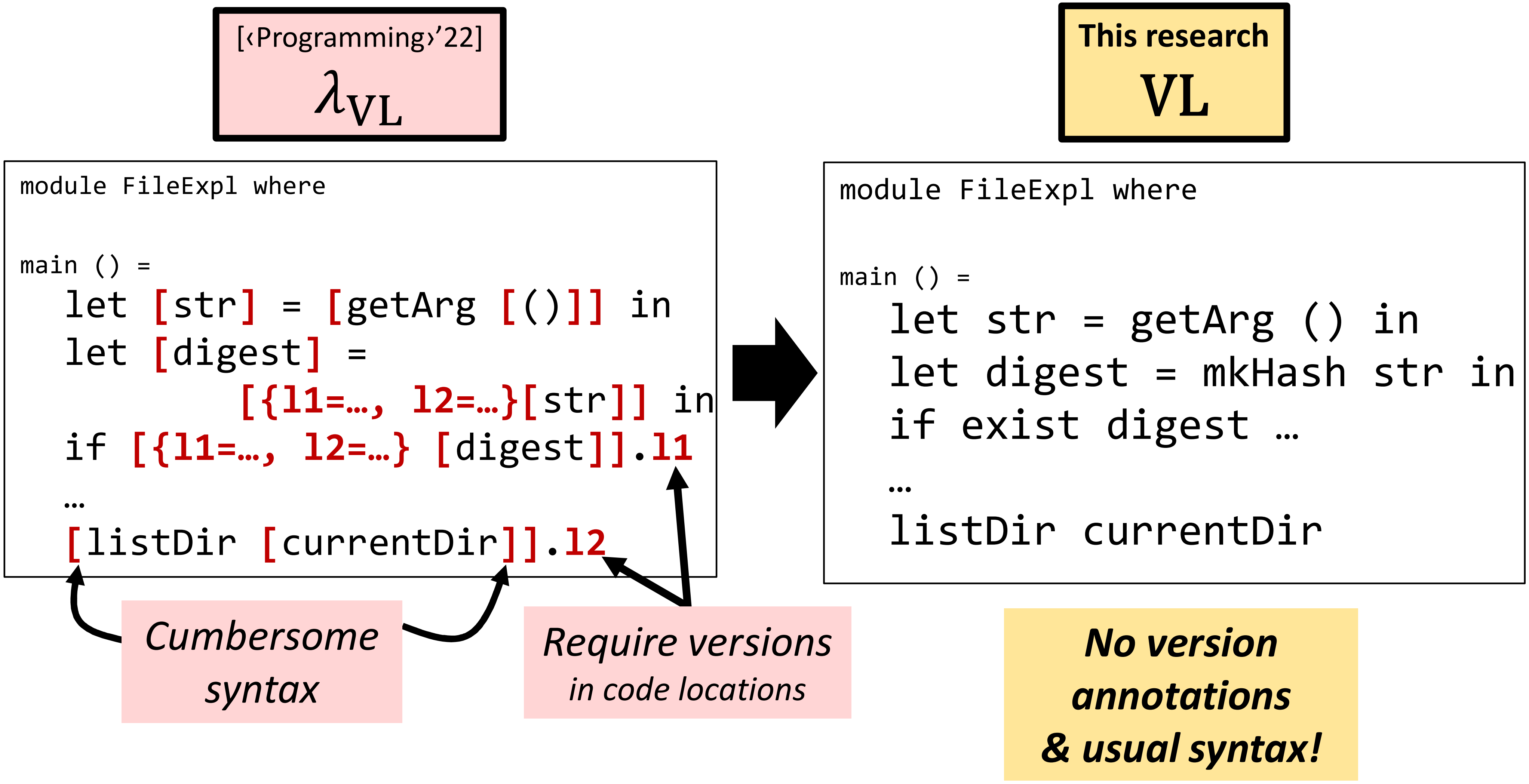


Basis of this research (next slide)

FP: λ_{VL}[‹Programming›'22], OOP: BatakJava[SLE'22]

Contribution

PwV *w/o* Version Annotations



[‹Programming›’22]

λ_{VL}

```
module FileExpl where

main () =
  let [str] = [getArg [()]] in
  let [digest] =
        [{l1=…, l2=…}[str]] in
  if [{l1=…, l2=…} [digest]].l1
  …
  [listDir [currentDir]].l2
```

Cumbersome syntax

Require versions in code locations

This research

VL

```
module FileExpl where

main () =
  let str = getArg () in
  let digest = mkHash str in
  if exist digest …
  …
  listDir currentDir
```

No version annotations & usual syntax!

Rest of the Talk

Contribution

Programming with Versions ***w/o*** Version Annotations

[‹Programming›'22]

λ_{VL}

Explicit version annotations

vs.

This research

VL

IR

VLMini

Version inference incorporating implicit versions

- λ_{VL} Semantics and Type System

- **Key idea**: ***Multi-version interface***
- VL Programming
- Compilation

- Implementation & Evaluation
- Future work

Outline

Contribution

Programming with Versions ***w/o*** Version Annotations

[‹Programming›'22]

λ_{VL}

Explicit
version annotations

vs.

This research

VL

IR

VLMini

Version inference
incorporating implicit versions

- **λ_{VL} Semantics and Type System**

- <u>Key idea</u>: ***Multi-version interface***
- VL Programming
- Compilation

- Implementation & Evaluation
- Future work

Background – [‹Programming›'22]

λ_{VL}, Versions within Semantics

Version Labels to capture multiple version possibilities

Multiple terms in a ***versioned value***

$$findFile = \left\{ \begin{array}{l} \boldsymbol{l_1} = \boxed{\begin{array}{l}\texttt{\textbackslash hash ->}\\ \texttt{ if exist hash …}\end{array}} \\ \boldsymbol{l_2} = \boxed{\begin{array}{l}\texttt{\textbackslash hash ->}\\ \texttt{ if exist hash …}\end{array}} \end{array} \right\}$$

Evaluate term in a specific version

$$[findFile\ hash].\boldsymbol{l_1}$$

$$\longrightarrow\ findFile_{l_1}\ hash_{l_1}$$

$$\longrightarrow\ \texttt{/home/yudaitnb/vl/src/file.ext}$$

$$l ::= [\overline{\underbrace{M_i}_{\textbf{\textit{Module name}}} \mapsto \underbrace{V_i}_{\textbf{\textit{Version number}}}}]$$

i.e.

$$l_1 = [Dir \mapsto 1.0.0]$$

$$l_2 = [Dir \mapsto 2.0.0,\ Hash \mapsto 1.0.0]$$

Background – [‹Programming›’22]

λ_{VL} Type System

Type system to ***enforce version consistency***

$$\begin{array}{l} \mathit{findFile} : \Box_{\{l_1\}}(\mathrm{Hash} \to A) \\ \mathit{mkHash} : \Box_{\underline{\{l_1,l_2\}}}\mathrm{Hash} \end{array} \vdash \begin{array}{l} \mathrm{let}\ [f] = \mathit{findFile}\ \mathrm{in} \\ \mathrm{let}\ [x] = \mathit{mkHash}\ \mathrm{in} \\ [f\ x].\boldsymbol{l_2} \end{array} : \begin{array}{c} \textbf{Well-} \\ \textbf{typed?} \end{array}$$

Types are tagged with

version resources

that denotes available versions of a term

Background – [‹Programming›'22]

λ_{VL} Type System

*Type system to **enforce version consistency***

$$\begin{array}{l} findFile : \Box_{\{l_1\}}(\text{Hash} \to A) \\ mkHash : \Box_{\{l_1,l_2\}}\text{Hash} \end{array} \vdash \begin{array}{l} \text{let } [f] = findFile \text{ in} \\ \text{let } [x] = mkHash \text{ in} \\ [f\ x]_{l_2} \end{array} : \text{Well-typed?}$$

$$: \Box_{\underline{\{l_1\}}} A$$

$$\|$$

$$\{l_1\} \cap \{l_1, l_2\}$$

Capture ***shared version resource***
to enforce consistent version usage

Background — [‹Programming›’22]

λ_{VL} Type System

*Type system to **enforce version consistency***

$$\begin{array}{l} findFile : \Box_{\{l_1\}}(\mathrm{Hash} \to A) \\ mkHash : \Box_{\{l_1, l_2\}}\mathrm{Hash} \end{array} \vdash \begin{array}{l} \mathrm{let}\ [f] = \underline{findFile}\ \mathrm{in} \\ \mathrm{let}\ [x] = mkHash\ \mathrm{in} \\ [f\ x].l_2 \end{array} :$$

~~Well-typed?~~

Type error

because $l_2 \notin \{l_1\} \cap \{l_1, l_2\}$

Background – [‹Programming›’22]

λ_{VL} Type System

Type system to ***enforce version consistency***

$$
\begin{array}{l}
findFile : \Box_{\{l_1\}}(\text{Hash} \to A) \\
mkHash : \Box_{\{l_1, l_2\}}\text{Hash}
\end{array}
\vdash
\begin{array}{l}
\text{let } [f] = \underline{findFile} \text{ in} \\
\text{let } [x] = mkHash \text{ in} \\
[f\ x].\, l_2
\end{array}
:
$$

~~Well-typed?~~

Type error

because $l_2 \notin \{l_1\} \cap \{l_1, l_2\}$

Proved

Type soundness

$\Gamma \vdash t : A \wedge t \longrightarrow t' \Longrightarrow \Gamma \vdash t' : A$ (preservation)

$\emptyset \vdash t : A \Longrightarrow \text{value } t \ \vee \exists t'.\, t \longrightarrow t'$ (progress)

Type system is based on coeffect calculi:

$\ell\mathcal{R}$PCF[Brunel’14], GrMini[Orchard’19].

Outline

Contribution

Programming with Versions ***w/o*** Version Annotations

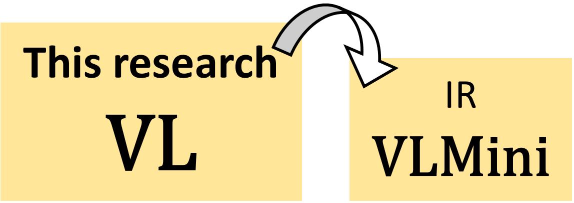
[‹Programming›'22]

λ_{VL}

Explicit version annotations

- λ_{VL} Semantics and Type System

vs.



This research

VL

IR

VLMini

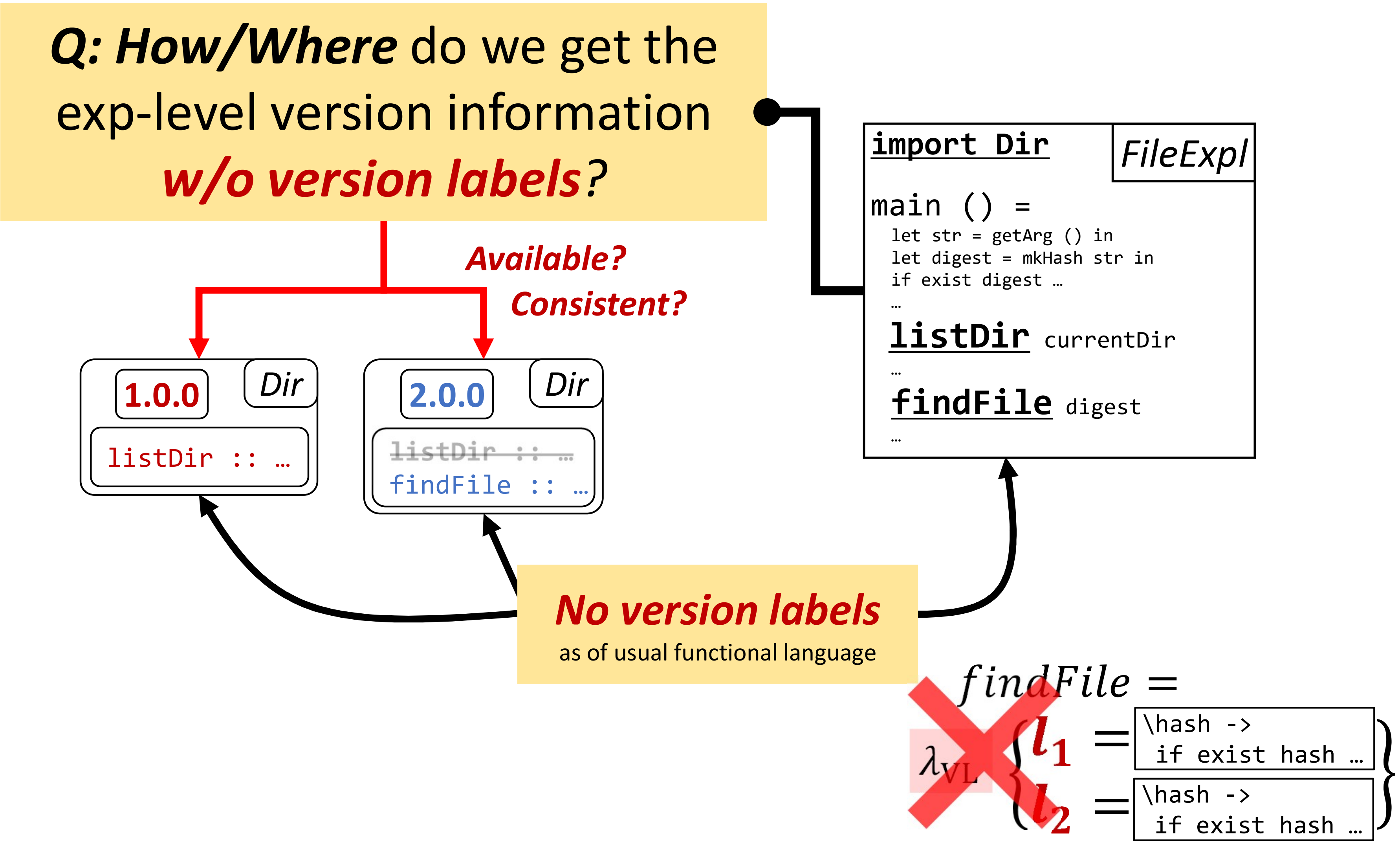
Version inference incorporating implicit versions

- **Key idea**: ***Multi-version interface***
- **VL Programming**
- **Compilation**

- Implementation & Evaluation
- Future work

Contribution

Version Inference *w/o* Version Annotations



Q: How/Where do we get the exp-level version information ***w/o version labels****?*

Available?

Consistent?

1.0.0 *Dir*

```
listDir :: …
```

2.0.0 *Dir*

```
~~listDir :: …~~
findFile :: …
```

FileExpl

```
import Dir

main () =
  let str = getArg () in
  let digest = mkHash str in
  if exist digest …
  …
  listDir currentDir
  …
  findFile digest
  …
```

No version labels

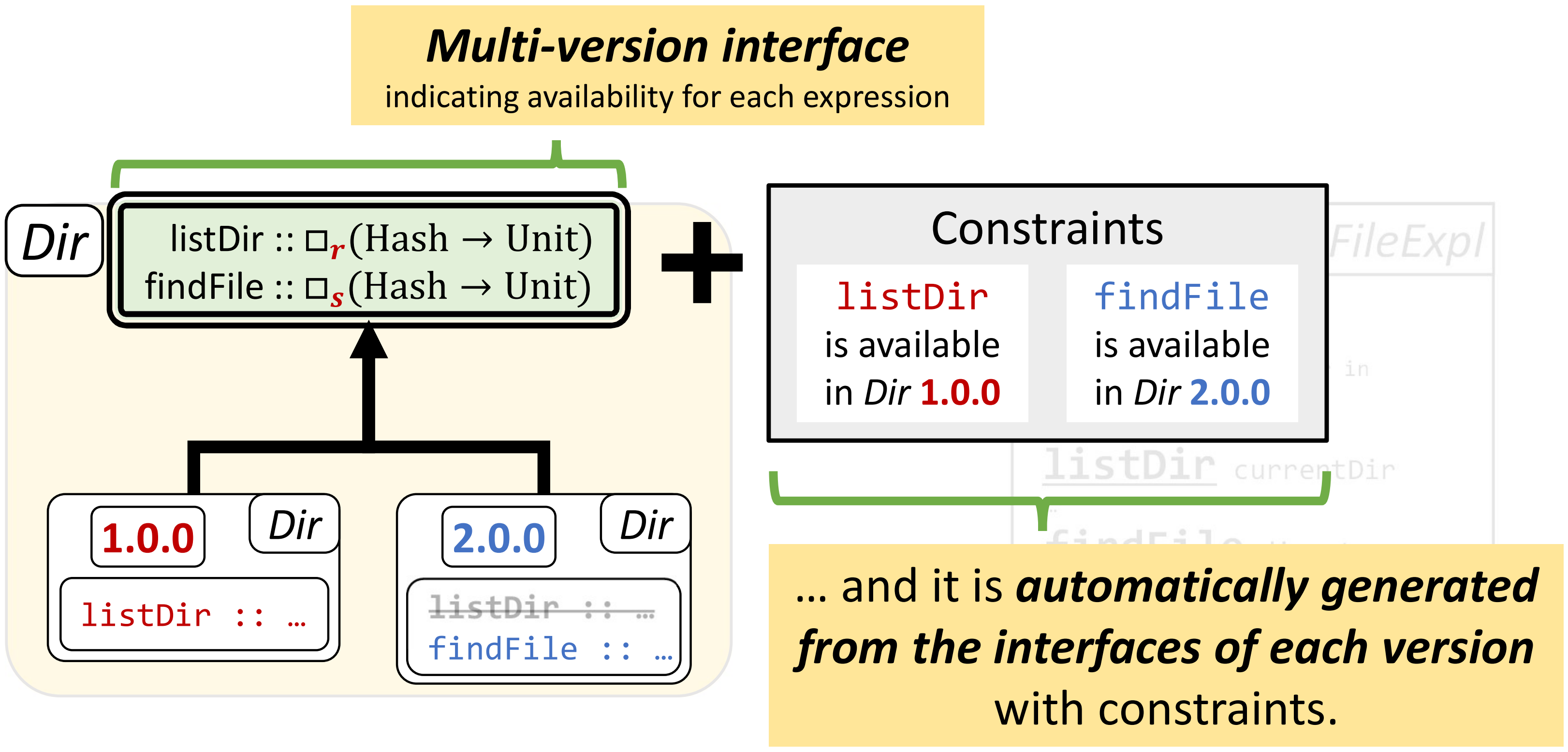
as of usual functional language

$$findFile = \lambda_{\mathrm{VL}}\left\{\begin{array}{l} l_1 = \texttt{\textbackslash hash -> if exist hash …} \\ l_2 = \texttt{\textbackslash hash -> if exist hash …} \end{array}\right\}$$

Contribution

Key Ideas

A:** Version tagging **only at module boundaries



Multi-version interface
indicating availability for each expression

listDir :: $\Box_r(\mathrm{Hash} \to \mathrm{Unit})$
findFile :: $\Box_s(\mathrm{Hash} \to \mathrm{Unit})$

Constraints

listDir is available in *Dir* **1.0.0**

findFile is available in *Dir* **2.0.0**

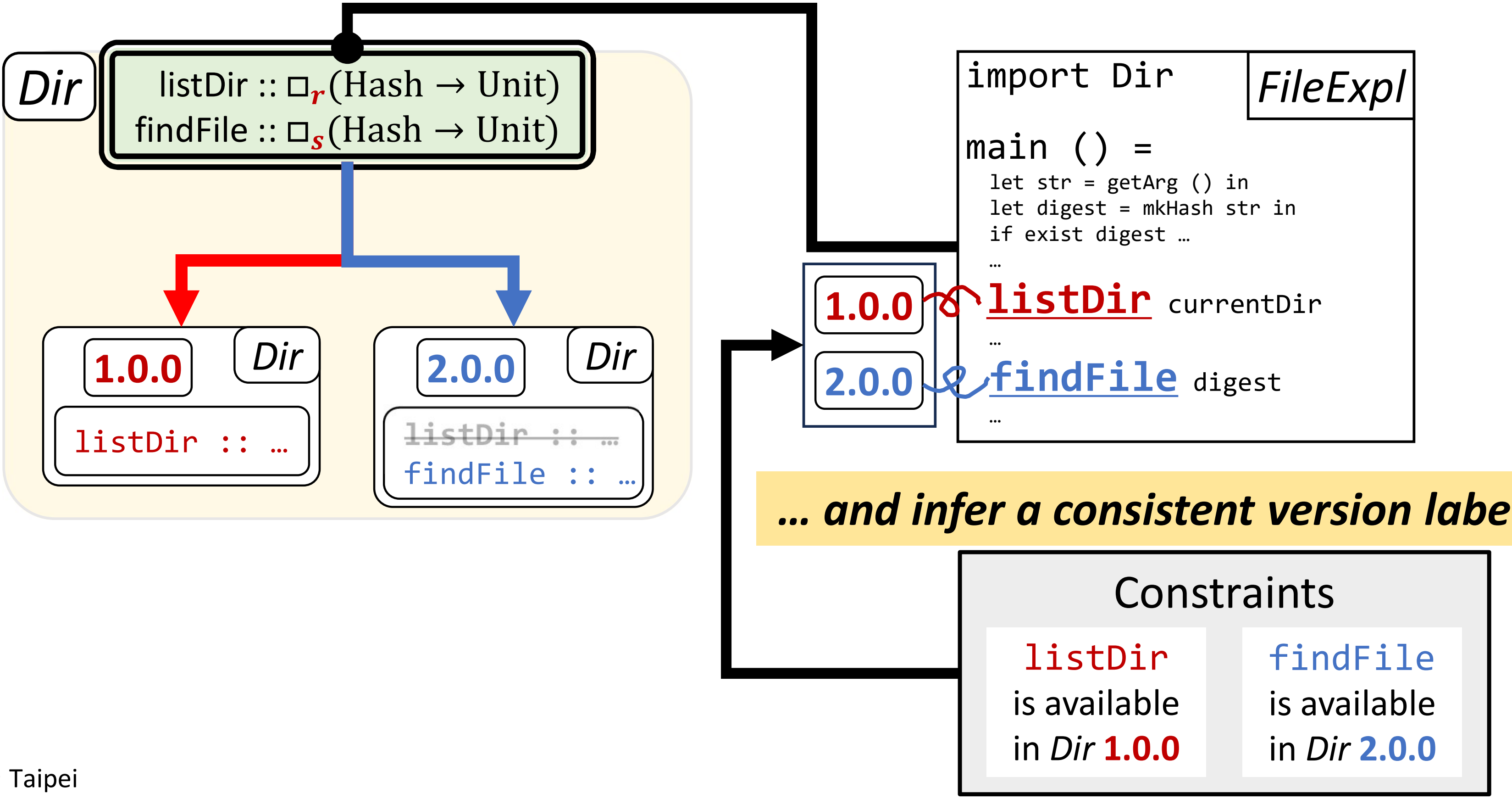
… and it is ***automatically generated from the interfaces of each version*** with constraints.

Contribution

Key Ideas

A: *Version tagging* ***only at module boundaries***

Importing external variables through
multi-version interface …



Dir

listDir :: $\square_r(\text{Hash} \to \text{Unit})$
findFile :: $\square_s(\text{Hash} \to \text{Unit})$

1.0.0 Dir

```
listDir :: …
```

2.0.0 Dir

```
~~listDir :: …~~
findFile :: …
```

FileExpl

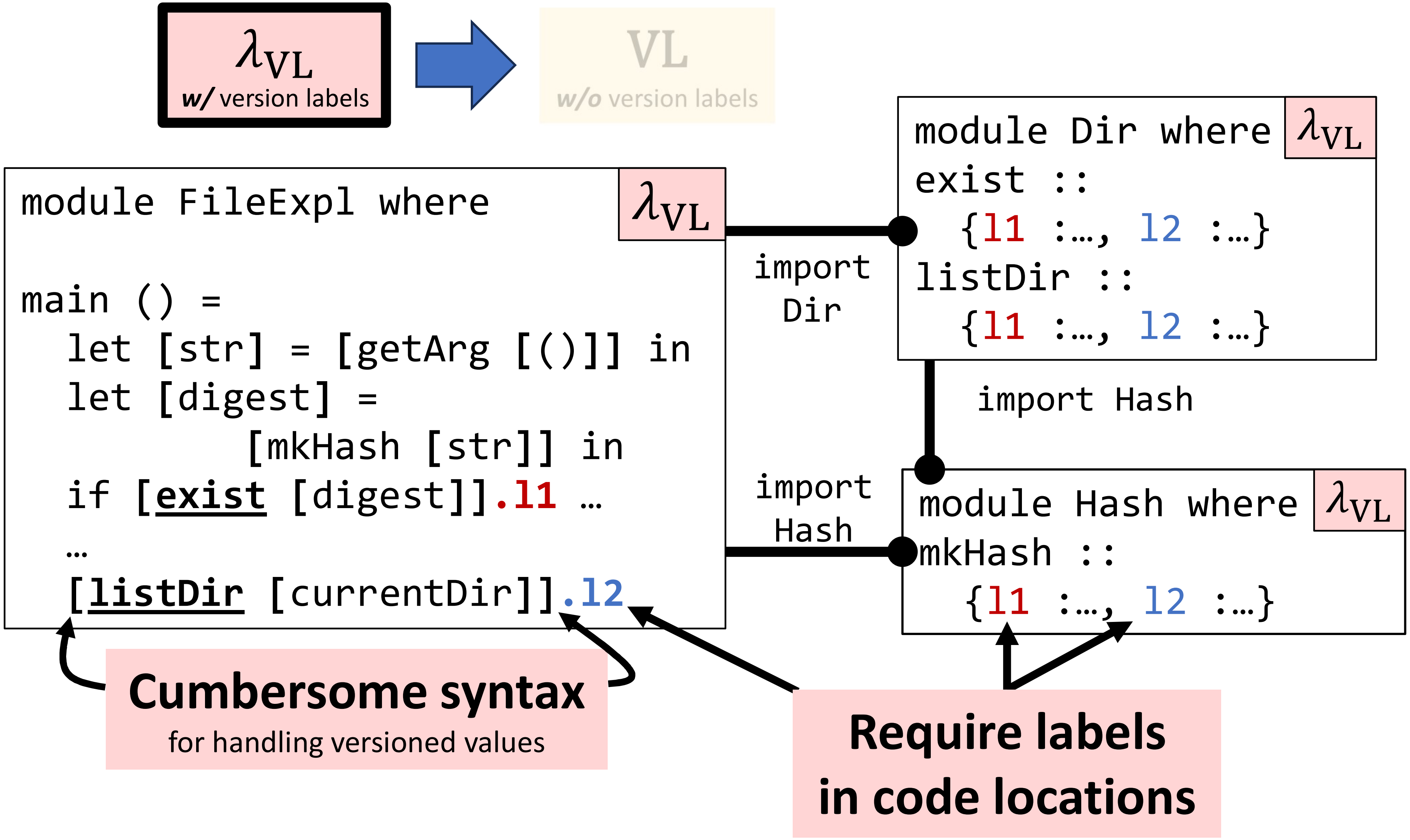
```
import Dir

main () =
  let str = getArg () in
  let digest = mkHash str in
  if exist digest …
  …
  listDir currentDir
  …
  findFile digest
  …
```

… and infer a consistent version label.

Constraints

listDir is available in *Dir* **1.0.0**	findFile is available in *Dir* **2.0.0**

VL *vs.* λ_{VL}



λ_{VL} ***w/*** version labels → **VL** ***w/o*** version labels

```
module FileExpl where                          λVL

main () =
  let [str] = [getArg [()]] in
  let [digest] =
              [mkHash [str]] in
  if [exist [digest]].l1 …
  …
  [listDir [currentDir]].l2
```

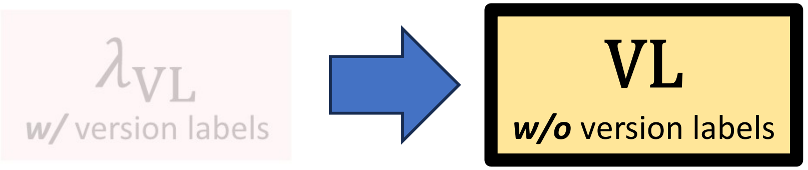
```
module Dir where                               λVL
exist ::
  {l1 :…, l2 :…}
listDir ::
  {l1 :…, l2 :…}
```

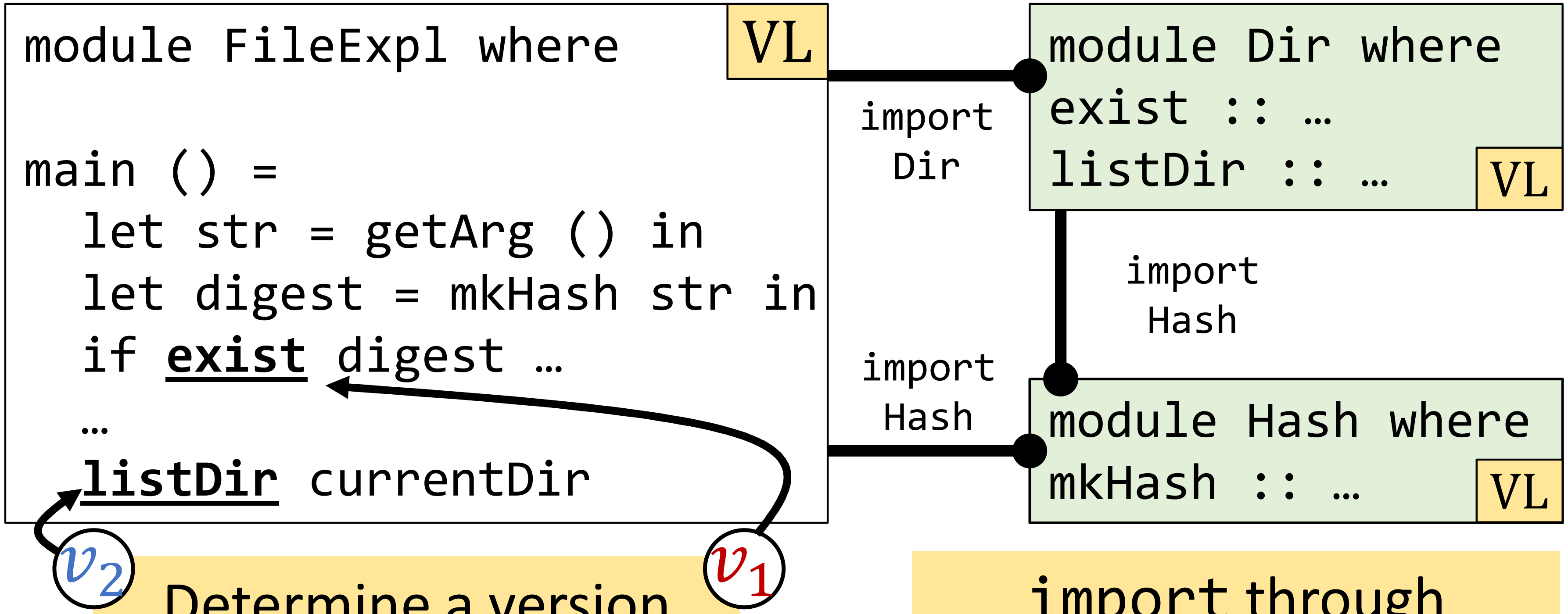
```
module Hash where                              λVL
mkHash ::
  {l1 :…, l2 :…}
```

Cumbersome syntax
for handling versioned values

Require labels in code locations

VL: Surface Language for λ_{VL}





```
module FileExpl where

main () =
  let str = getArg () in
  let digest = mkHash str in
  if exist digest …
  …
  listDir currentDir
```

```
module Dir where
exist :: …
listDir :: …
```

```
module Hash where
mkHash :: …
```

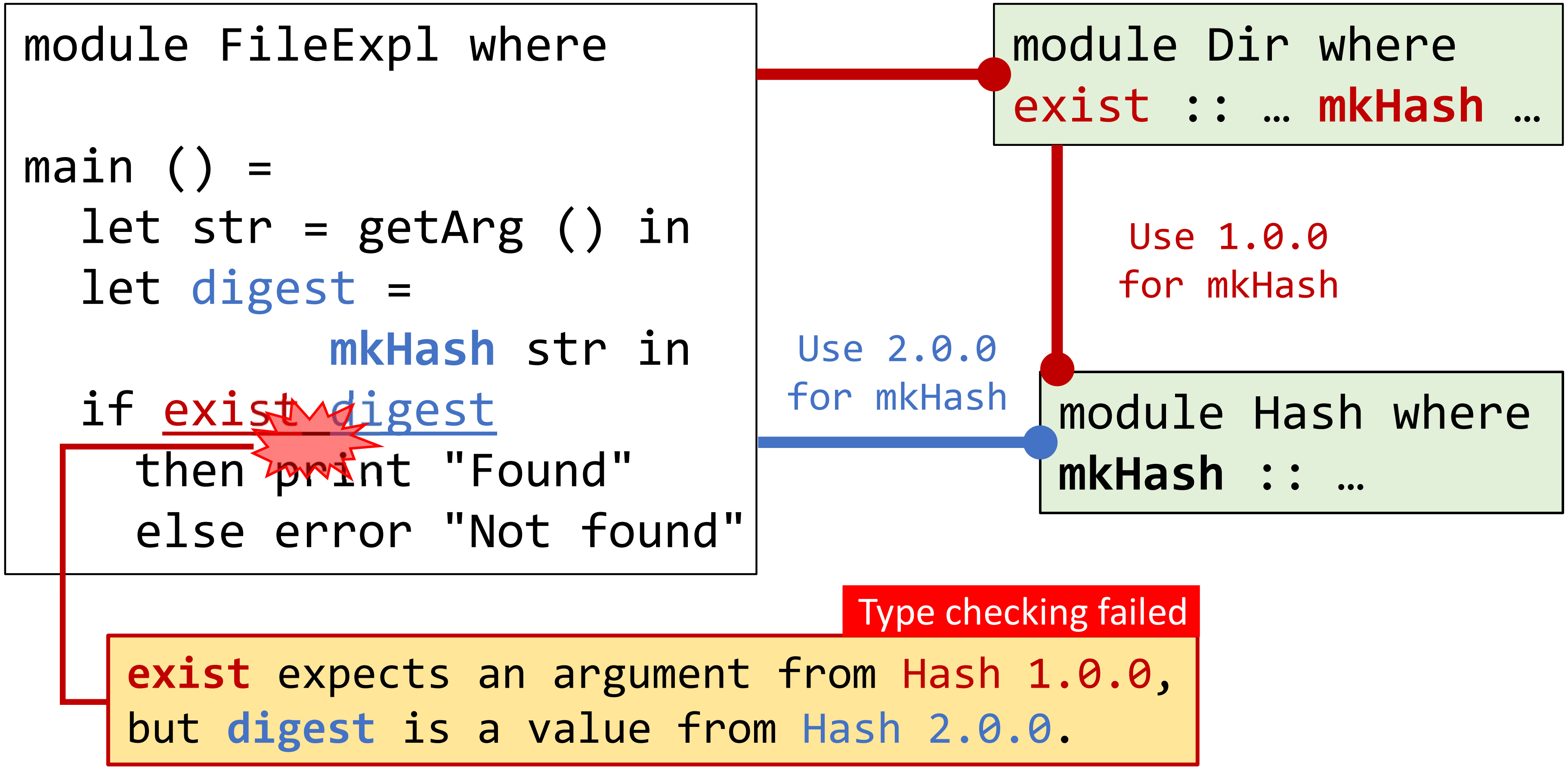
Determine a version
by semantic analysis

import through
multi-version interfaces

VL Type Checking

Goal

- Handling multiple versions in one client
- ***Detecting incompatible version usage***



```
module FileExpl where

main () =
  let str = getArg () in
  let digest =
           mkHash str in
  if exist digest
    then print "Found"
    else error "Not found"
```

```
module Dir where
exist :: … mkHash …
```

```
module Hash where
mkHash :: …
```

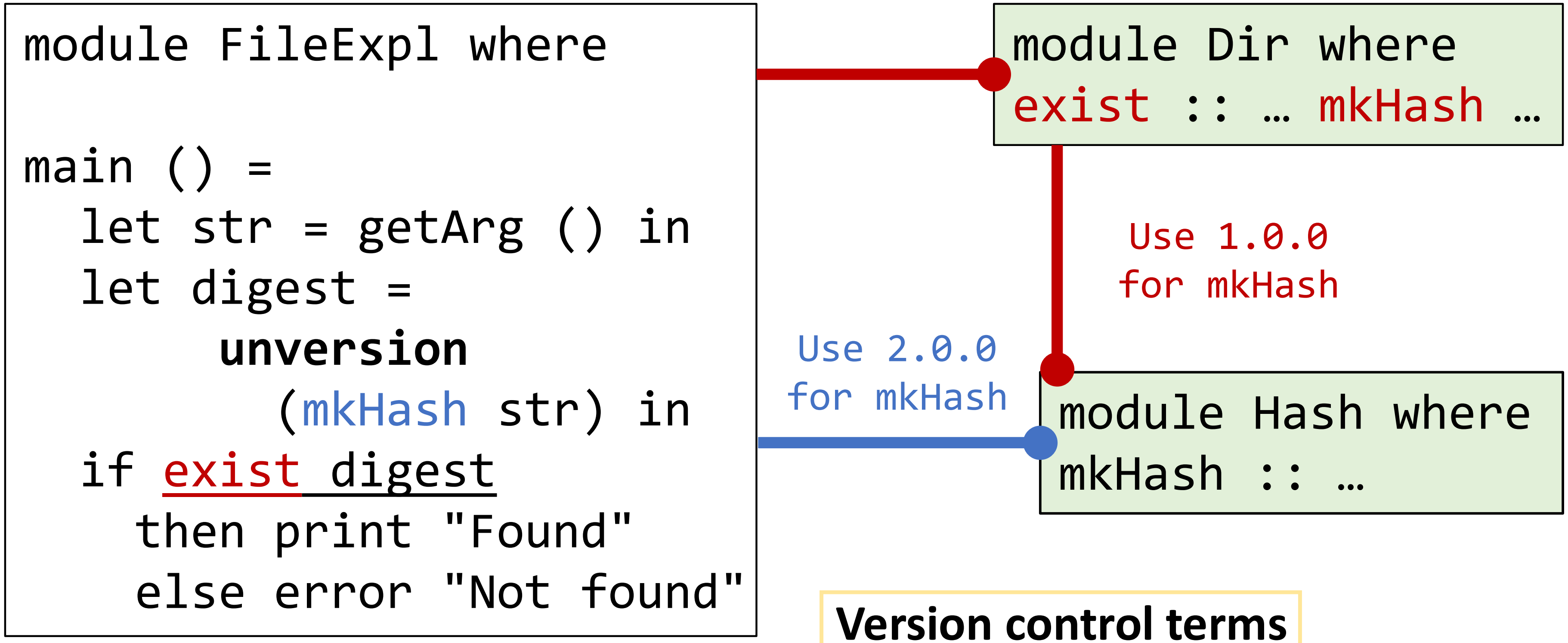
Type checking failed

```
exist expects an argument from Hash 1.0.0,
but digest is a value from Hash 2.0.0.
```

Version Control Terms

Goal

- ***Handling multiple versions in one client***
- Detecting incompatible version usage



```
module FileExpl where

main () =
  let str = getArg () in
  let digest =
        unversion
          (mkHash str) in
  if exist digest
   then print "Found"
   else error "Not found"
```

```
module Dir where
exist :: … mkHash …
```

```
module Hash where
mkHash :: …
```

Version control terms

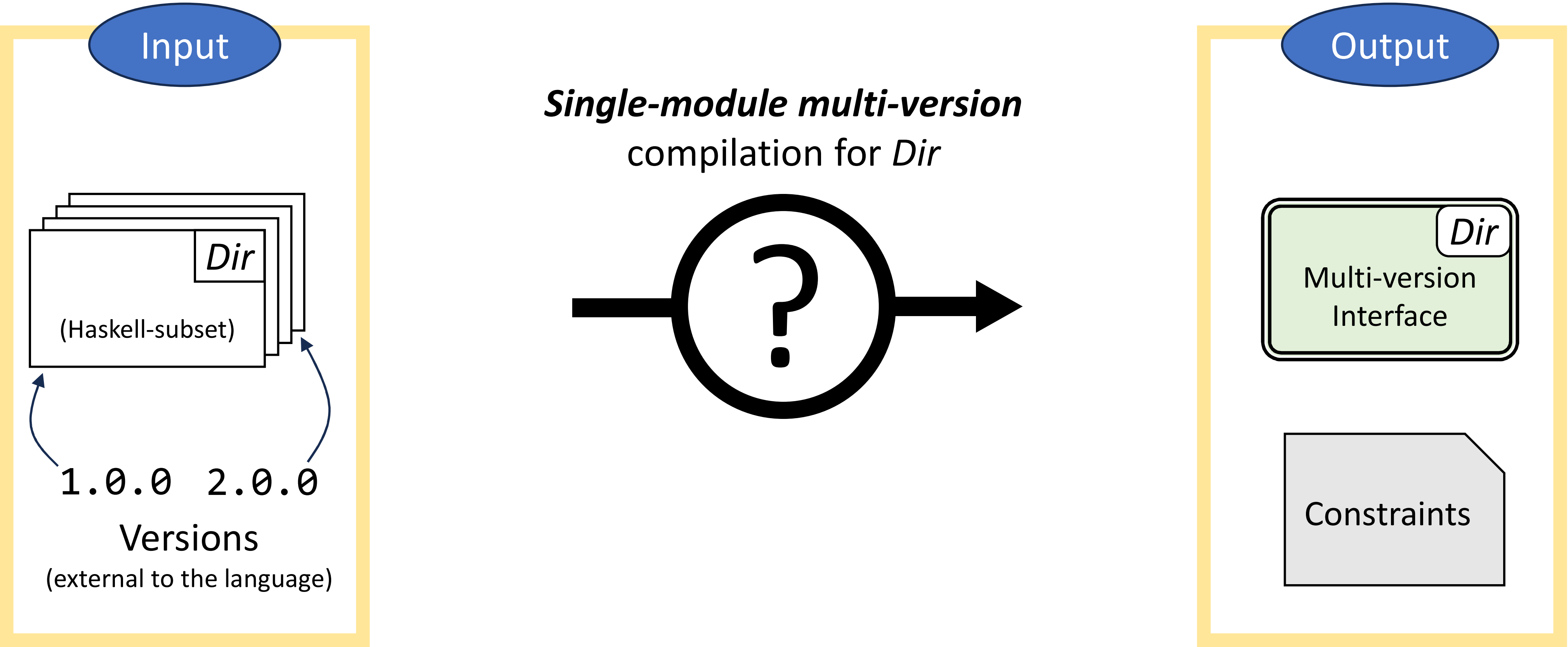
- **unversion** $\boldsymbol{t}$ eases t's constraints
- **version** $\{\overline{\boldsymbol{M_i \mapsto V_i}}\}$ **of** $\boldsymbol{t}$ specifies t's versions

Compilation Overview

In-/Out-put of Compilation

Contribution

Programming with versions
w/o version annotations



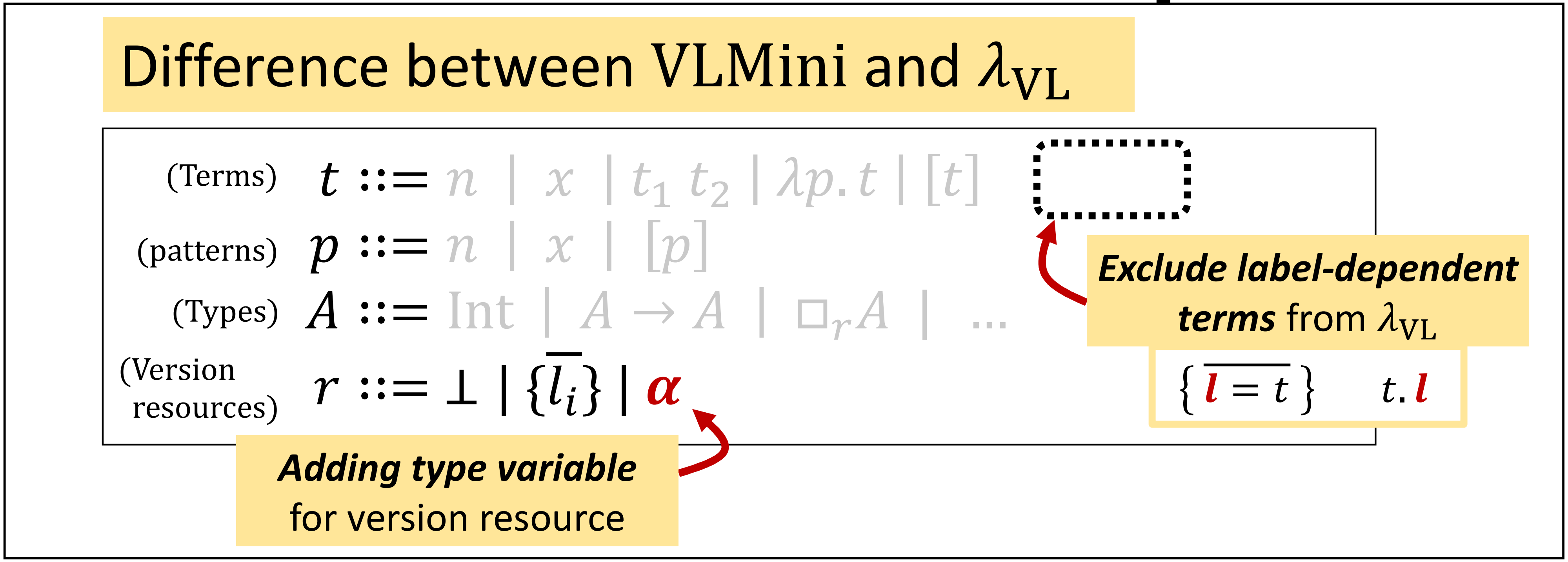
Compilation Overview

IR: VLMini

Programming with versions ***w/o*** version annotations *via*

IR: VLMini

A ***version-label-free*** variant of λ_{VL}



Difference between VLMini and λ_{VL}

(Terms) $t ::= n \mid x \mid t_1\, t_2 \mid \lambda p.t \mid [t]$

(patterns) $p ::= n \mid x \mid [p]$

(Types) $A ::= \mathrm{Int} \mid A \rightarrow A \mid \square_r A \mid \ldots$

(Version resources) $r ::= \bot \mid \{\overline{l_i}\} \mid \alpha$

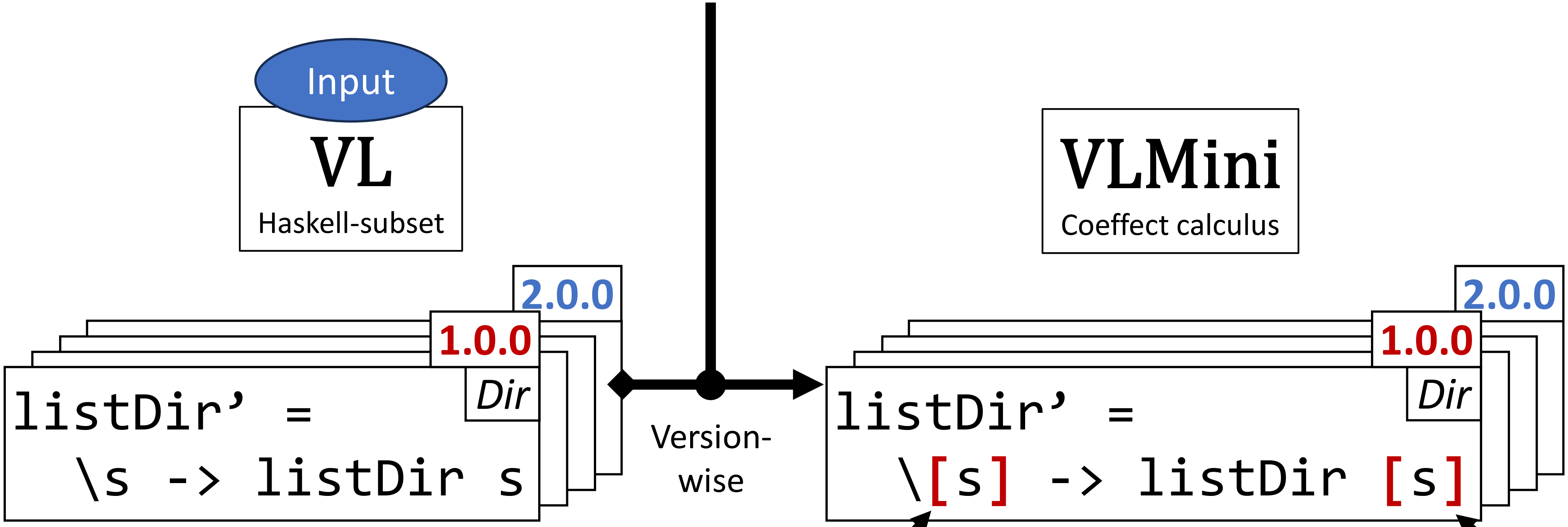
Exclude label-dependent terms from λ_{VL}

$\{\overline{l = t}\}$ $\quad t.l$

Adding type variable for version resource

Compilation Overview

① Inserting Annotations



```
listDir’ =
  \s -> listDir s
```

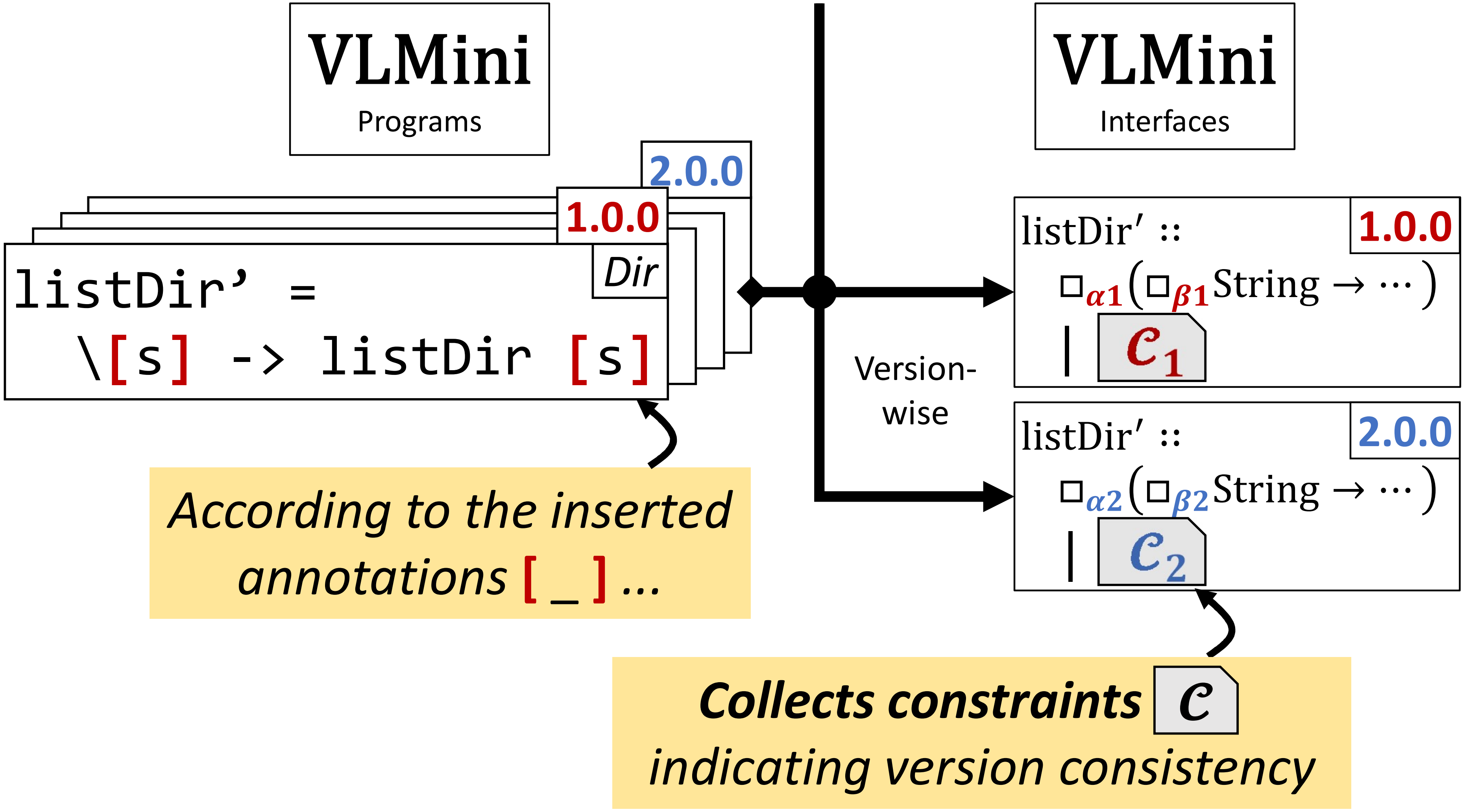
```
listDir’ =
  \[s] -> listDir [s]
```

Treat arguments as versioned values …

… and ***capture their version resources** by promoted patterns*

Compilation Overview

② Version Inference



VLMini
Programs

2.0.0

1.0.0

Dir

```
listDir' =
  \[s] -> listDir [s]
```

According to the inserted annotations [_] …

Version-wise

VLMini
Interfaces

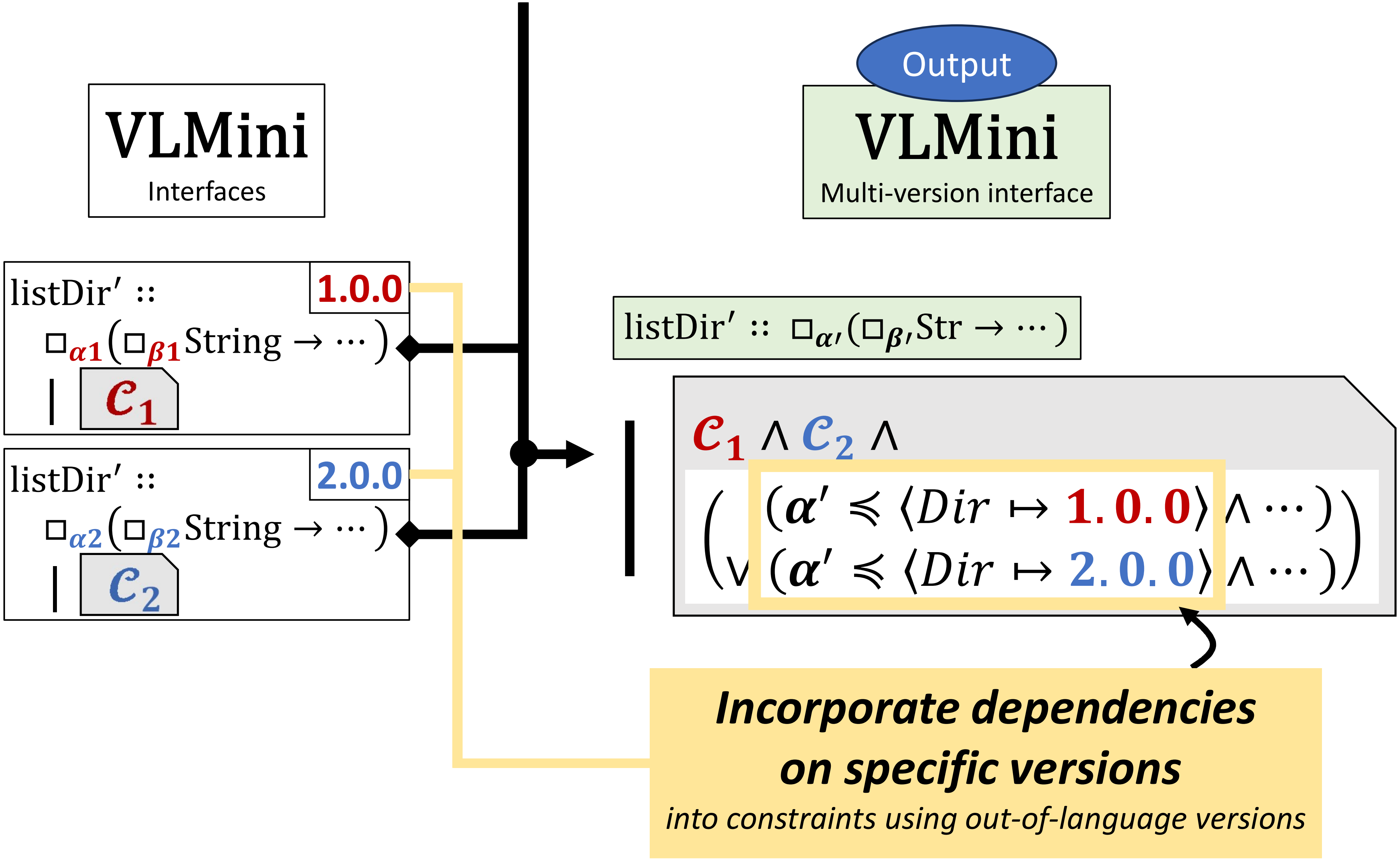
listDir′ :: **1.0.0**
$\square_{\alpha 1}(\square_{\beta 1}\mathrm{String} \to \cdots)$
$\mid \mathcal{C}_1$

listDir′ :: **2.0.0**
$\square_{\alpha 2}(\square_{\beta 2}\mathrm{String} \to \cdots)$
$\mid \mathcal{C}_2$

Collects constraints $\mathcal{C}$ *indicating version consistency*

Compilation Overview

③ Interface Bundling



VLMini

Interfaces

Output

VLMini

Multi-version interface

listDir′ :: $\square_{\alpha 1}(\square_{\beta 1}\text{String} \to \cdots)$ | $\mathcal{C}_1$ **1.0.0**

listDir′ :: $\square_{\alpha 2}(\square_{\beta 2}\text{String} \to \cdots)$ | $\mathcal{C}_2$ **2.0.0**

listDir′ :: $\square_{\alpha'}(\square_{\beta'}\text{Str} \to \cdots)$

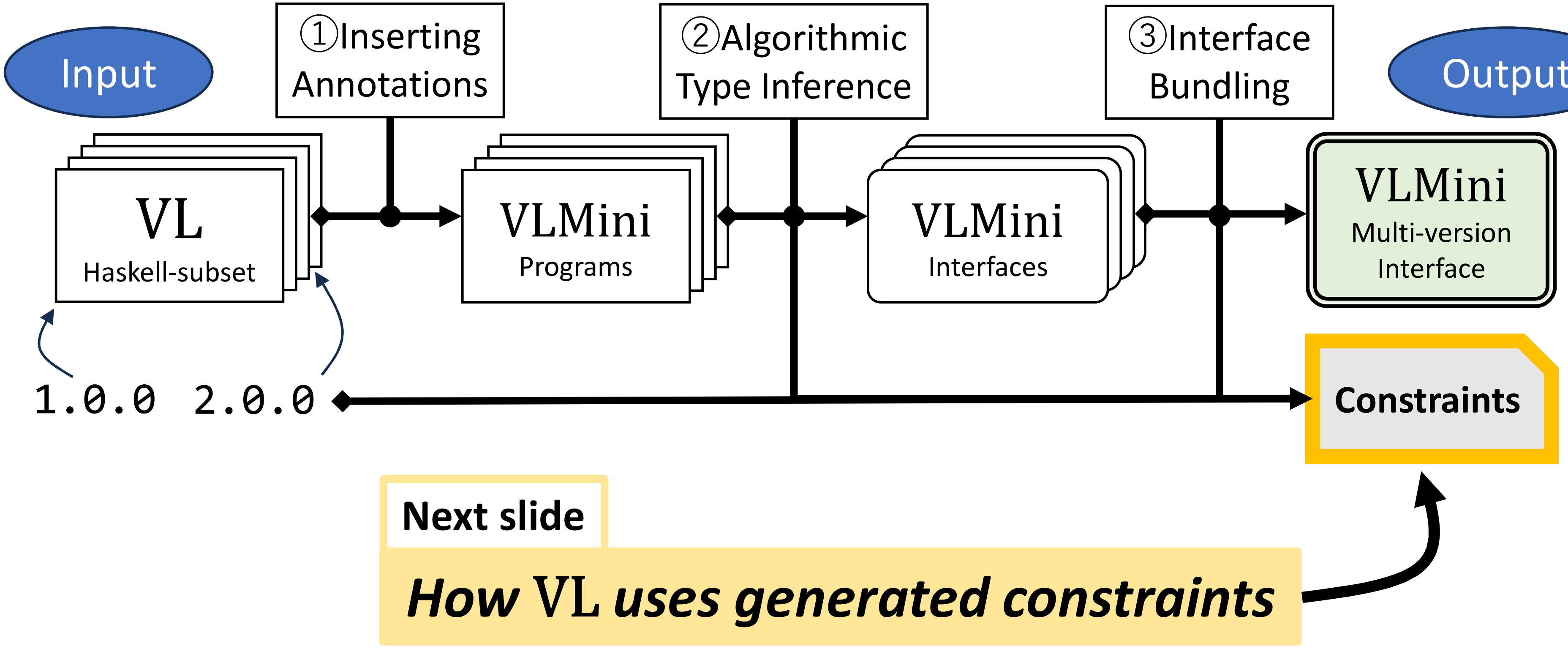
$$\Big|\ \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \left(\begin{array}{l} (\alpha' \preccurlyeq \langle Dir \mapsto \mathbf{1.0.0} \rangle \wedge \cdots) \\ \vee\ (\alpha' \preccurlyeq \langle Dir \mapsto \mathbf{2.0.0} \rangle \wedge \cdots) \end{array}\right)$$

Incorporate dependencies on specific versions

into constraints using out-of-language versions

Compilation Overview

In-/Out-put of Compilation



The **VL** Core Language

Constraints

“≼” representing dependencies

(Constraints) $\mathcal{C} ::= \top \mid \mathcal{C}_1 \wedge \mathcal{C}_2 \mid \mathcal{C}_1 \vee \mathcal{C}_2 \mid \boxed{\alpha \preccurlyeq \alpha' \mid \alpha \preccurlyeq \mathcal{D}}$

(Dependencies) $\mathcal{D} ::= \langle \overline{\underbrace{M_i}_{\text{Module name}} \mapsto \underbrace{V_i}_{\text{Version number}}} \rangle$

*“If a version label for **RHS** expects a specific version, …*

$$\underbrace{\alpha}_{\text{LHS}} \preccurlyeq \overbrace{\alpha'}^{\text{RHS}} \qquad \underbrace{\alpha}_{\text{LHS}} \preccurlyeq \overbrace{\langle Dir \mapsto 1.0.0 \rangle}^{\text{RHS}}$$

*… then **α** (LHS) also expects the same version.”*

Satisfiable Constraints

Constraints

$$\begin{cases} \alpha \preccurlyeq \alpha' \\ \alpha' \preccurlyeq \langle Dir \mapsto 1.0.0 \rangle \end{cases}$$

+

$$\text{listDir} : \square_{\alpha} A$$

Constraint resolution

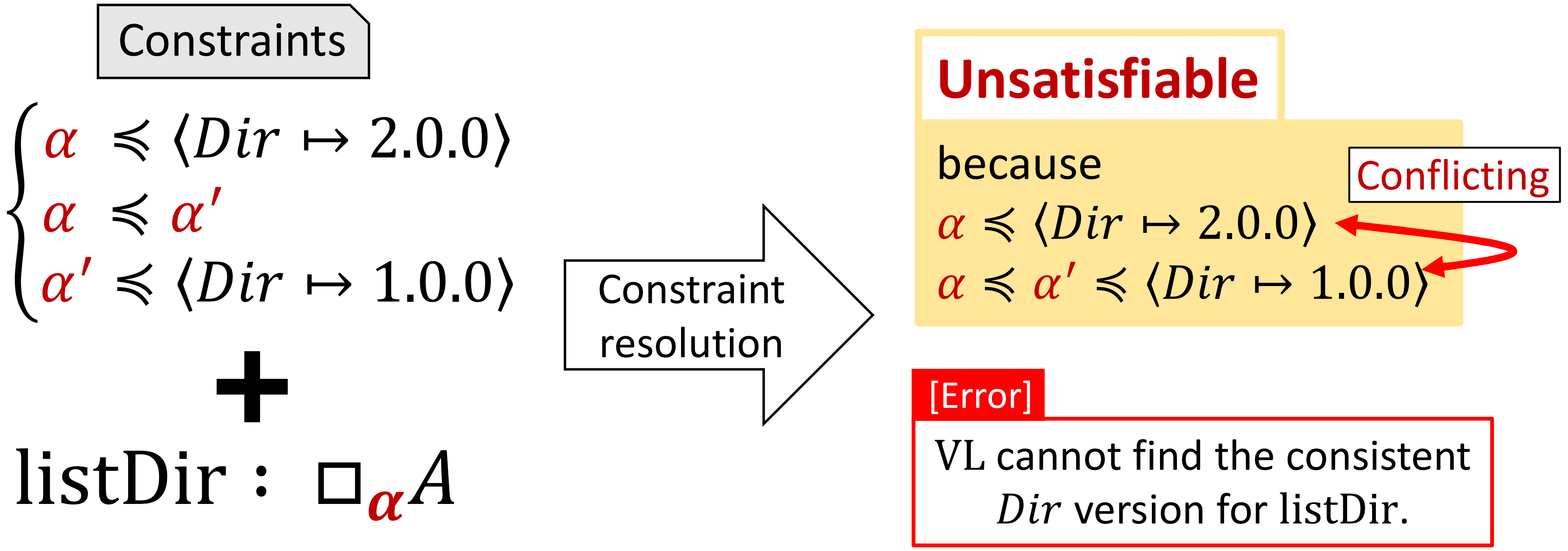
Satisfiable

$$\alpha = \alpha' = [Dir \mapsto 1.0.0]$$

✓ listDir is compiled using version 1.0.0 of *Dir.*

Constraint Resolution

Unsatisfiable Constraints



Constraints

$$\begin{cases} \alpha \preccurlyeq \langle Dir \mapsto 2.0.0 \rangle \\ \alpha \preccurlyeq \alpha' \\ \alpha' \preccurlyeq \langle Dir \mapsto 1.0.0 \rangle \end{cases}$$

$$+$$

$$\text{listDir} : \Box_{\alpha} A$$

Constraint resolution

Unsatisfiable

because

$\alpha \preccurlyeq \langle Dir \mapsto 2.0.0 \rangle$

$\alpha \preccurlyeq \alpha' \preccurlyeq \langle Dir \mapsto 1.0.0 \rangle$

Conflicting

[Error]

VL cannot find the consistent Dir version for listDir.

Outline

Contribution

Programming with Versions ***w/o*** Version Annotations

[‹Programming›'22]

λ_{VL}

Explicit version annotations

- λ_{VL} Semantics and Type System

vs.

This research

VL

IR

VLMini

Version inference incorporating implicit versions

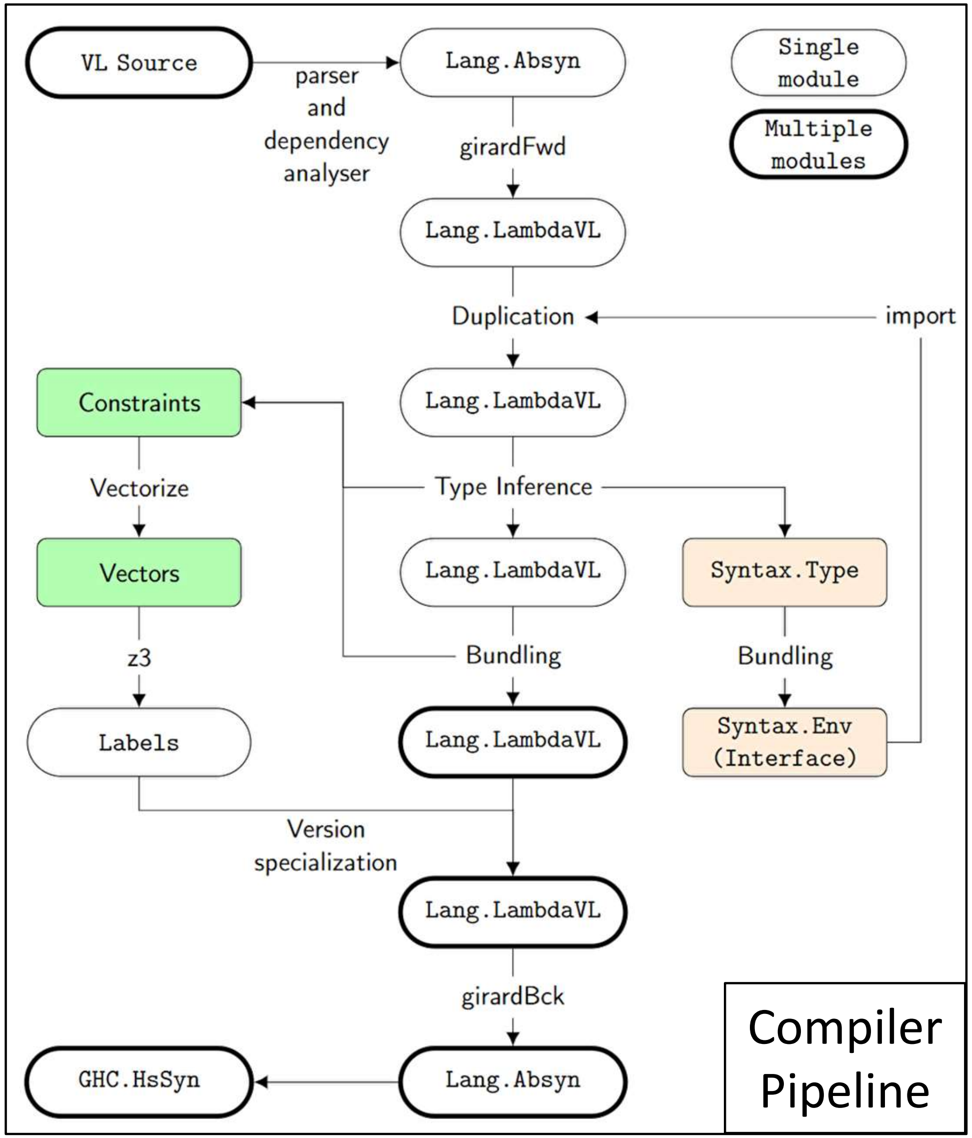
- **Key idea**: ***Multi-version interface***
- VL Programming
- Compilation

- **Implementation & Evaluation**
- **Future work**

Implementation

The VL Compiler

- Implemented on
  GHC 9.2.4
  https://github.com/yudaitnb/vl

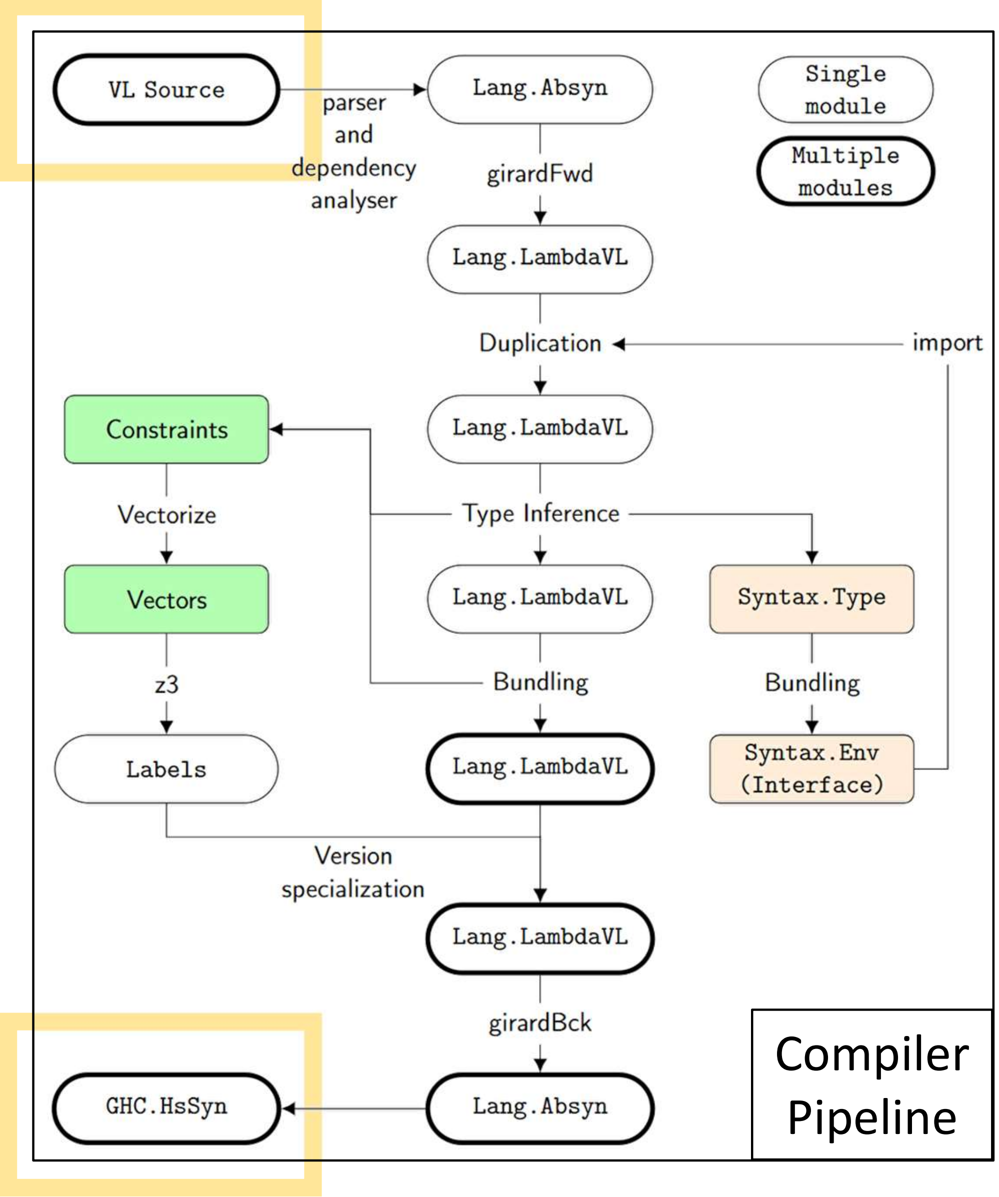


Implementation

The VL Compiler

- Implemented on GHC 9.2.4 https://github.com/yudaitnb/vl

- Input and output are Haskell ASTs



Compiler Pipeline

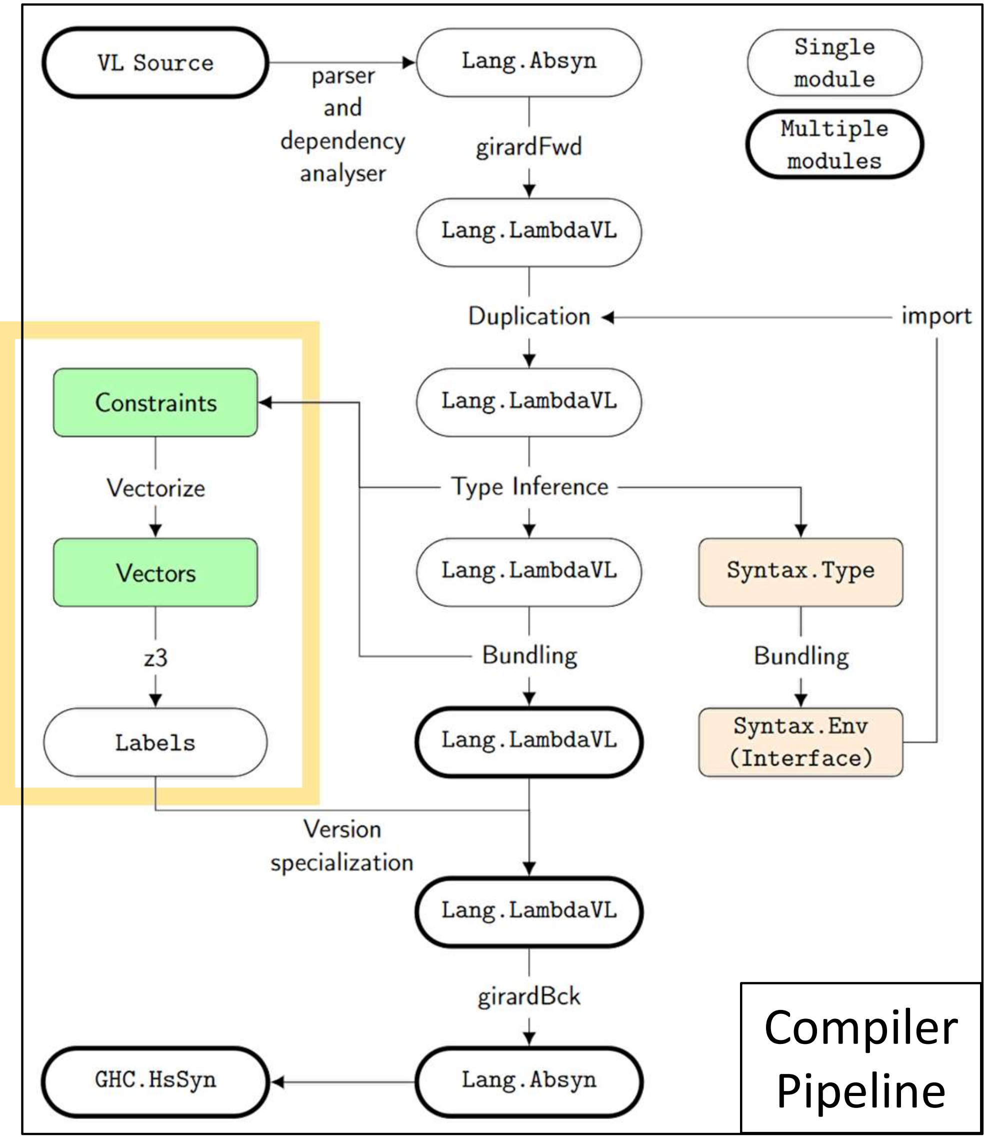
Implementation

The VL Compiler

- Implemented on GHC 9.2.4
  https://github.com/yudaitnb/vl

- Input and output are Haskell ASTs

- Resolve constraints using Z3 [De Moura'08]

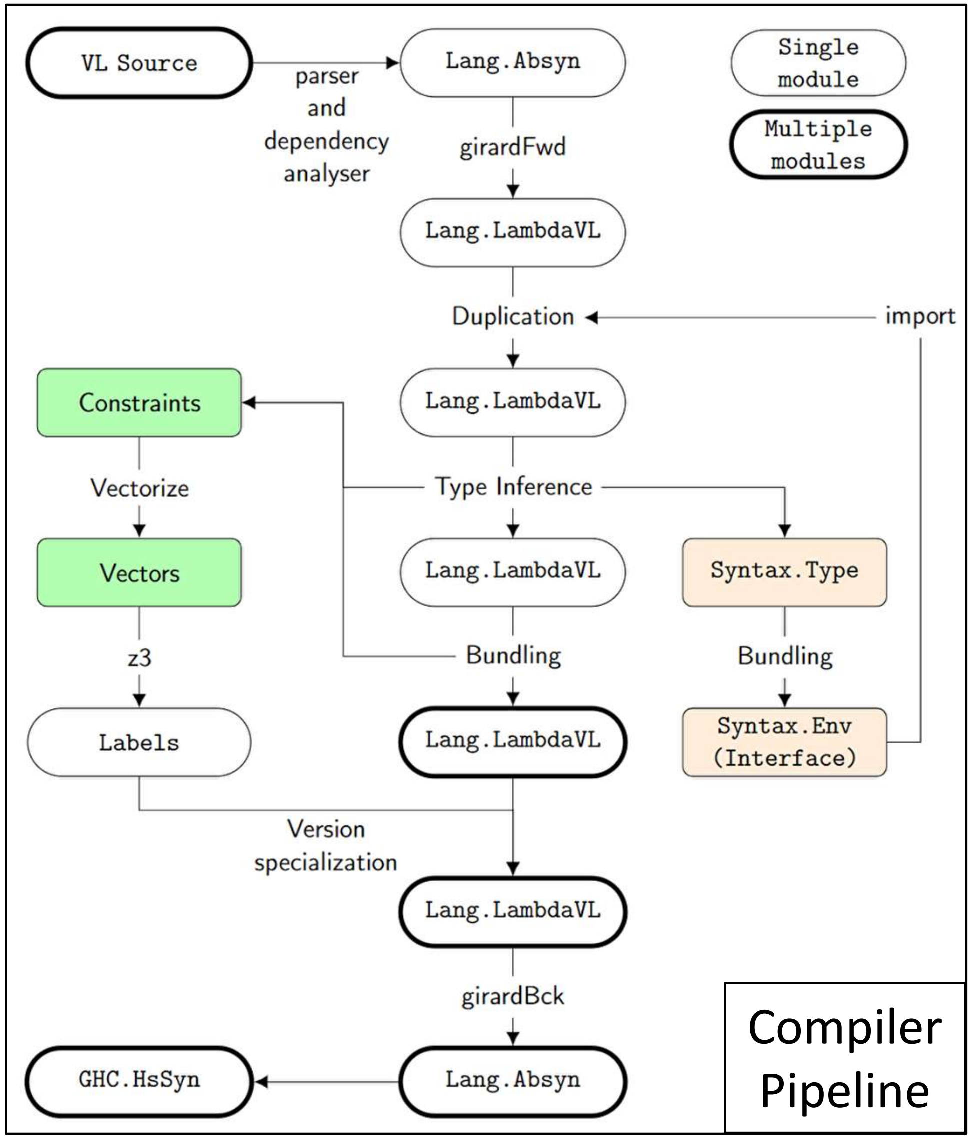


Compiler Pipeline

Implementation

The VL Compiler

- Implemented on GHC 9.2.4
  https://github.com/yudaitnb/vl
- Both in-/out-put are Haskell ASTs (subset)
- Resolve constraints using Z3 [De Moura'08]



Compiler Pipeline

Evaluations (next slides)

1. ***Case study*** to confirm VL achieving PwV benefits
2. ***Compiler performance***

Evaluation

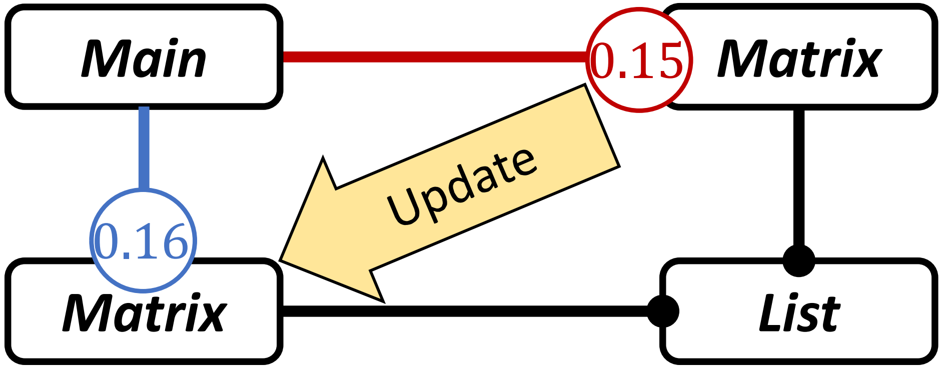
1. Case Study

VL ***achieves our goals:***

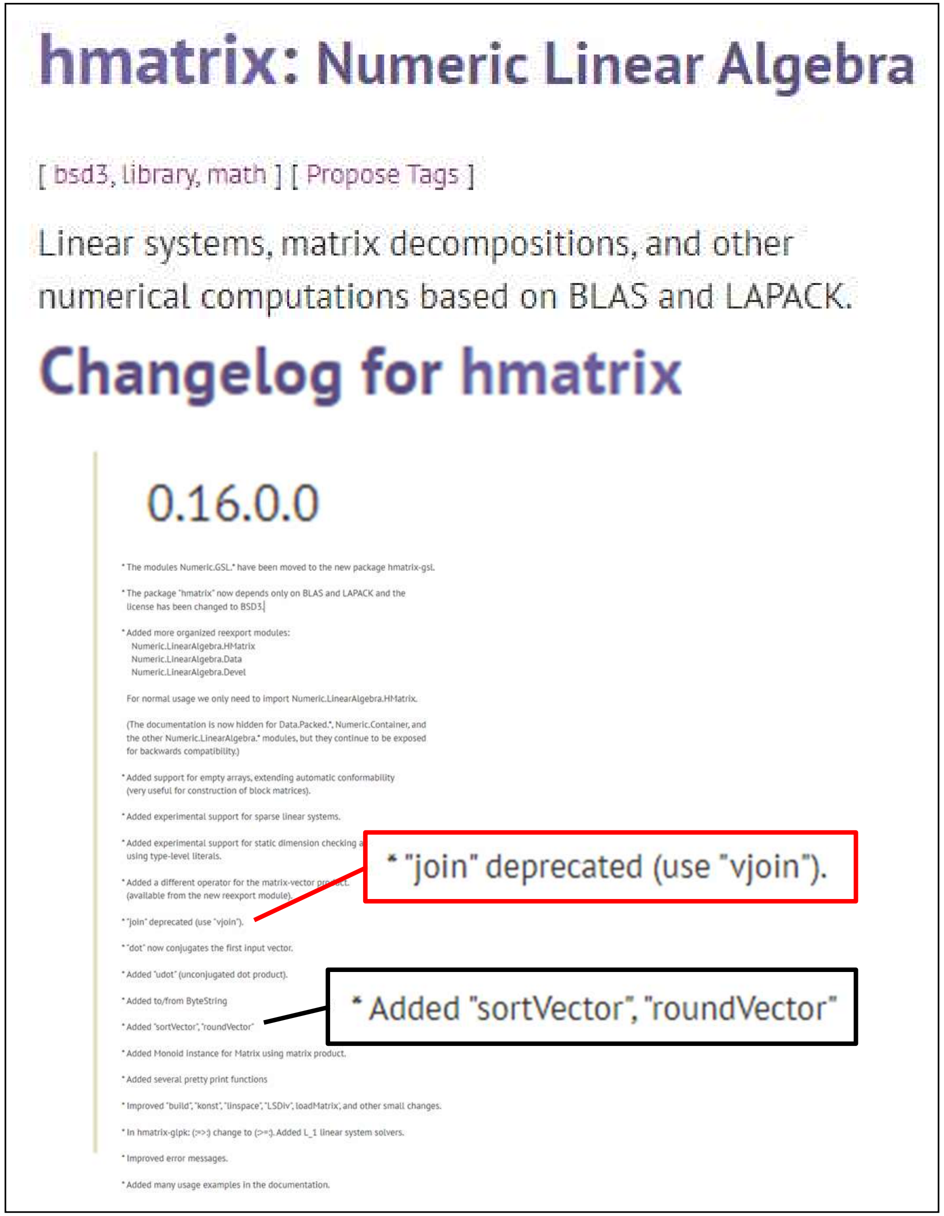
✓ *Handling two versions* in one client
✓ *Detecting inconsistent version*

Setting

- Port **hmatrix** to ***Matrix***
- Simulating breaking updates in VL



≈ Data.List
base (Haskell standard library)



hmatrix: Numeric Linear Algebra

[bsd3, library, math] [Propose Tags]

Linear systems, matrix decompositions, and other numerical computations based on BLAS and LAPACK.

Changelog for hmatrix

0.16.0.0

- The modules Numeric.GSL.* have been moved to the new package hmatrix-gsl.
- The package "hmatrix" now depends only on BLAS and LAPACK and the license has been changed to BSD3.
- Added more organized reexport modules:
  Numeric.LinearAlgebra.HMatrix
  Numeric.LinearAlgebra.Data
  Numeric.LinearAlgebra.Devel

  For normal usage we only need to import Numeric.LinearAlgebra.HMatrix.

  (The documentation is now hidden for Data.Packed.*, Numeric.Container, and the other Numeric.LinearAlgebra.* modules, but they continue to be exposed for backwards compatibility.)
- Added support for empty arrays, extending automatic conformability (very useful for construction of block matrices).
- Added experimental support for sparse linear systems.
- Added experimental support for static dimension checking a using type-level literals.
- Added a different operator for the matrix-vector product. (available from the new reexport module).
- "join" deprecated (use "vjoin").
- "dot" now conjugates the first input vector.
- Added "udot" (unconjugated dot product).
- Added to/from ByteString
- Added "sortVector", "roundVector"
- Added Monoid instance for Matrix using matrix product.
- Added several pretty print functions
- Improved "build", "konst", "linspace", "LSDiv", loadMatrix", and other small changes.
- In hmatrix-glpk: (:=>:) change to (:>=:). Added L_1 linear system solvers.
- Improved error messages.
- Added many usage examples in the documentation.

* "join" deprecated (use "vjoin").

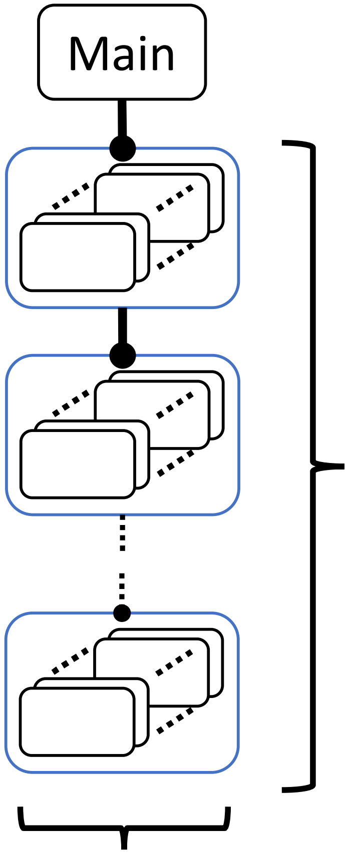
* Added "sortVector", "roundVector"

https://hackage.haskell.org/package/hmatrix-0.20.2
https://hackage.haskell.org/package/hmatrix-0.20.2/changelog

2. Compiler Performance

Benchmark setting

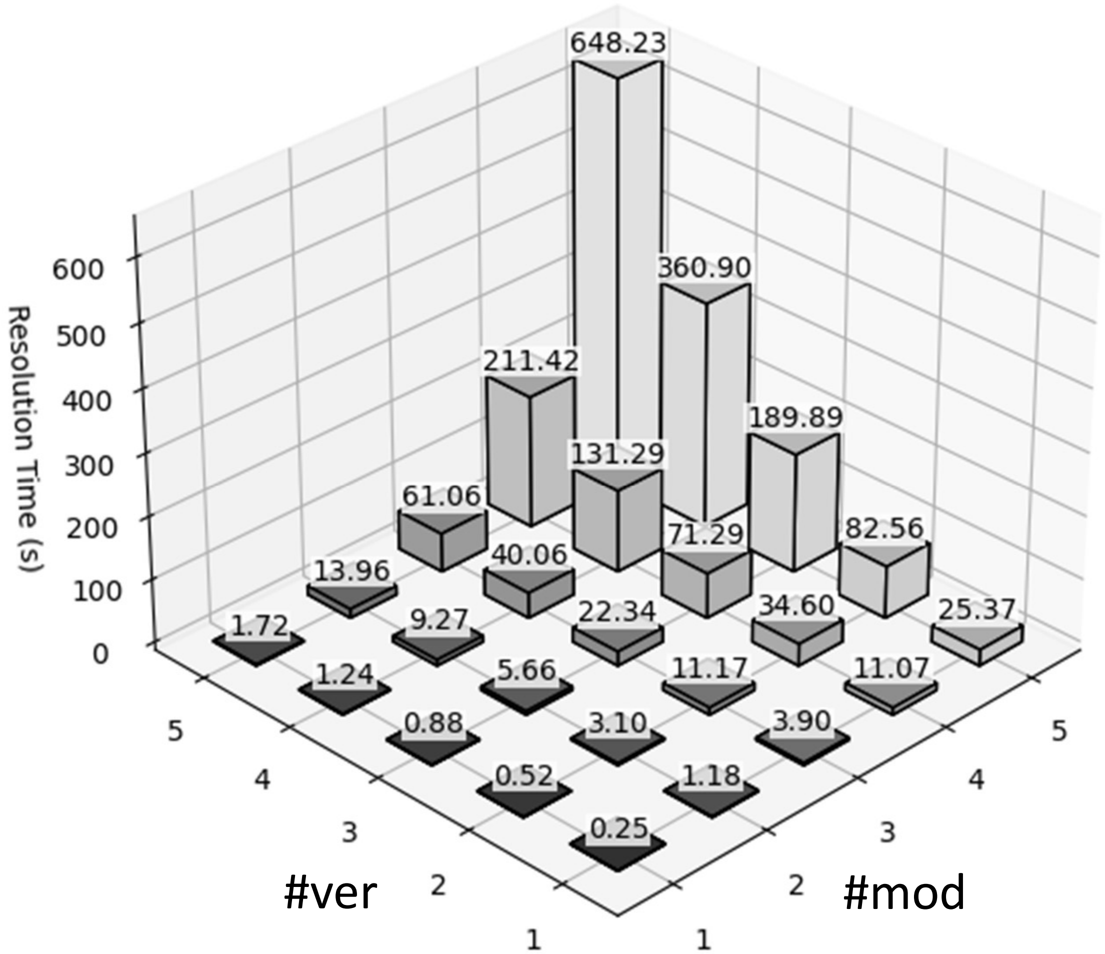
Ubuntu 22.04
Ryzen 7950X
Z3 version 4.12.2



Importing **#mod**-times nested dependencies

Each module has **#ver** versions

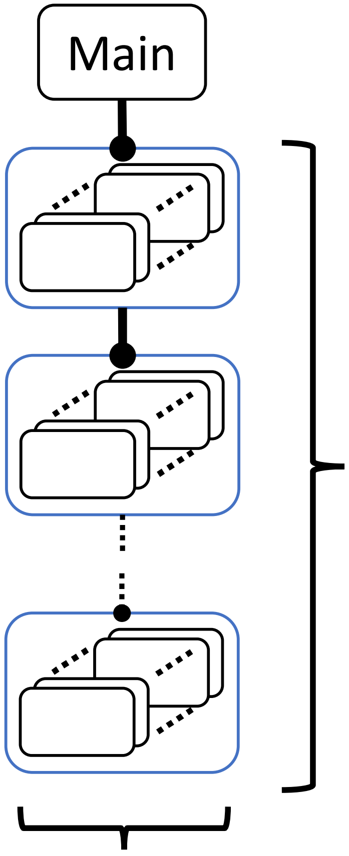
~500LOC
/ mod · ver



Evaluation

2. Compiler Performance

Benchmark setting

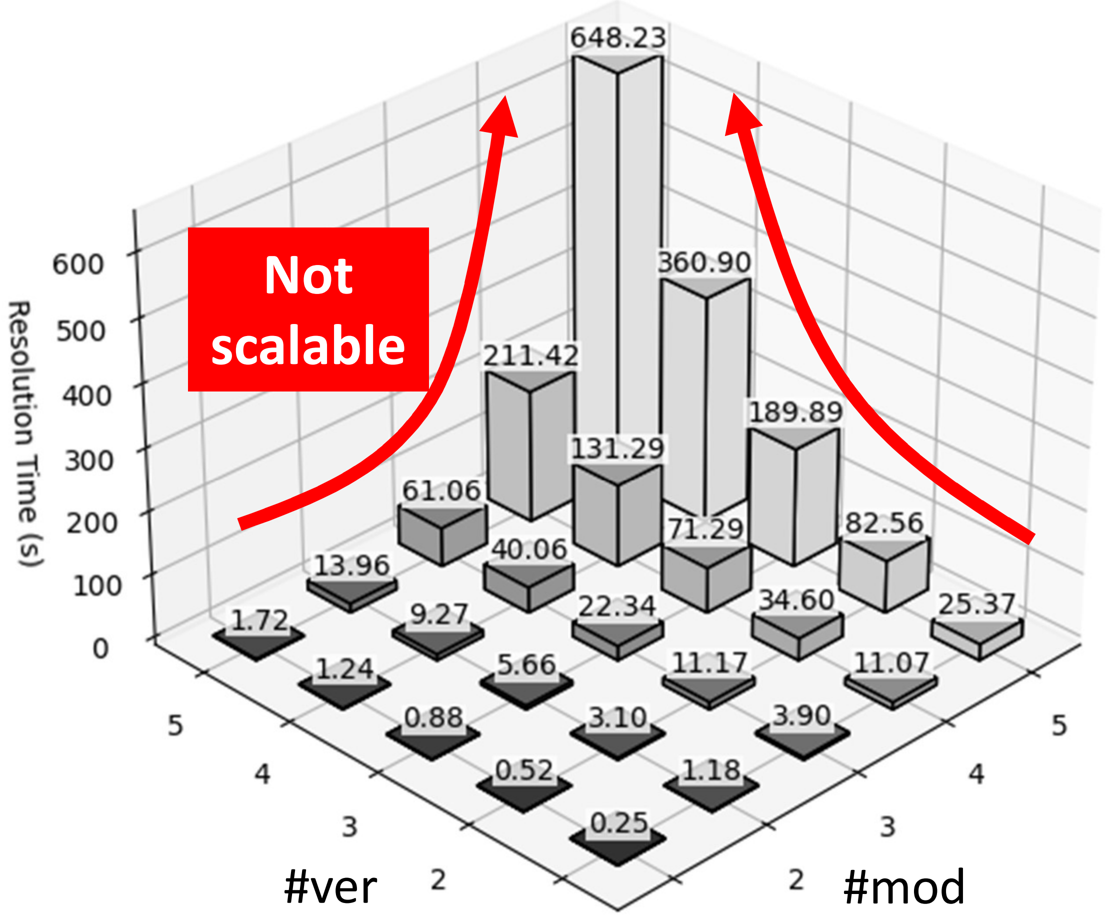


Ubuntu 22.04
Ryzen 7950X
Z3 version 4.12.2

Importing **#mod**-times nested dependencies

Each module has **#ver** versions

~500LOC / mod · ver



Not scalable

… but existing techniques can optimize the constraint resolution

(out-of-scope, short discussion in the paper)

Future Work

Further Compatibility Support

Supported changes

- Add/delete definitions
- Semantically *incompatible* changes
  (w/o interface-level changes)
- Add/Delete imports
  (w/o cyclic dependencies)

BatakJava [SLE'22] {
- Add/delete methods
- Class inheritance changes

Unsupported changes

- **Type changes**
- Semantically *compatible* changes

(Currently unsupported features)

- Data types
- Type classes
- License
- Visibility

Future Work

Further Compatibility Support

Supported **changes**

- Add/delete definitions
- Semantically *incompatible* changes
  (w/o interface-level changes)
- Add/Delete imports
  (w/o cyclic dependencies)

BatakJava [SLE'22] {
- Add/delete methods
- Class inheritance changes

Unsupported **changes**

- **Type changes**
- Semantically *compatible* changes

(Currently unsupported features)

- Data types
- Type classes
- License
- Visibility

Idea: ***Integrating PwV into record calculus***[Ohori'95]

$\boxed{\lambda_{\mathrm{VL}}}$? $\boxed{\lambda^{let,\cdot}}$

$$f : \Box_{\{l_1,l_2\}} A \;\simeq\; f : \forall t :: \langle l_1 : A, l_2 : B \rangle \,.\, t$$

Allow different types across versions unlike λ_{VL}

Summary

Contribution

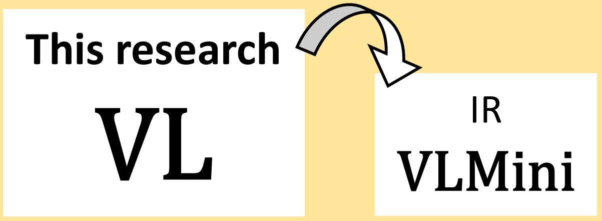
Programming with versions
w/o version annotations

[‹Programming›’22]

λ_{VL}

Explicit
version annotations

vs.


This research

VL

IR

VLMini

Version inference *using*
Multi-version interface
incorporating implicit versions

Implementation

on GHC with Z3
https://github.com/yudaitnb/vl

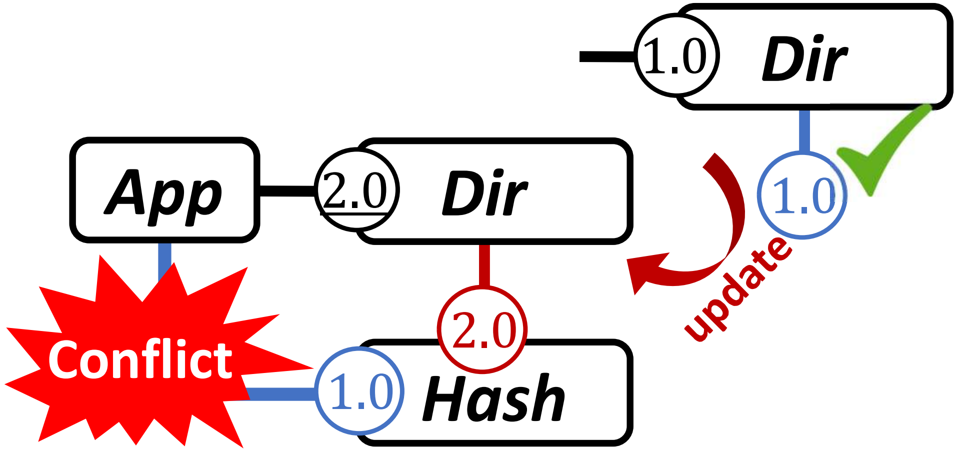
Preprint

Formalization
Proof of soundness

Background

Dependency Hell

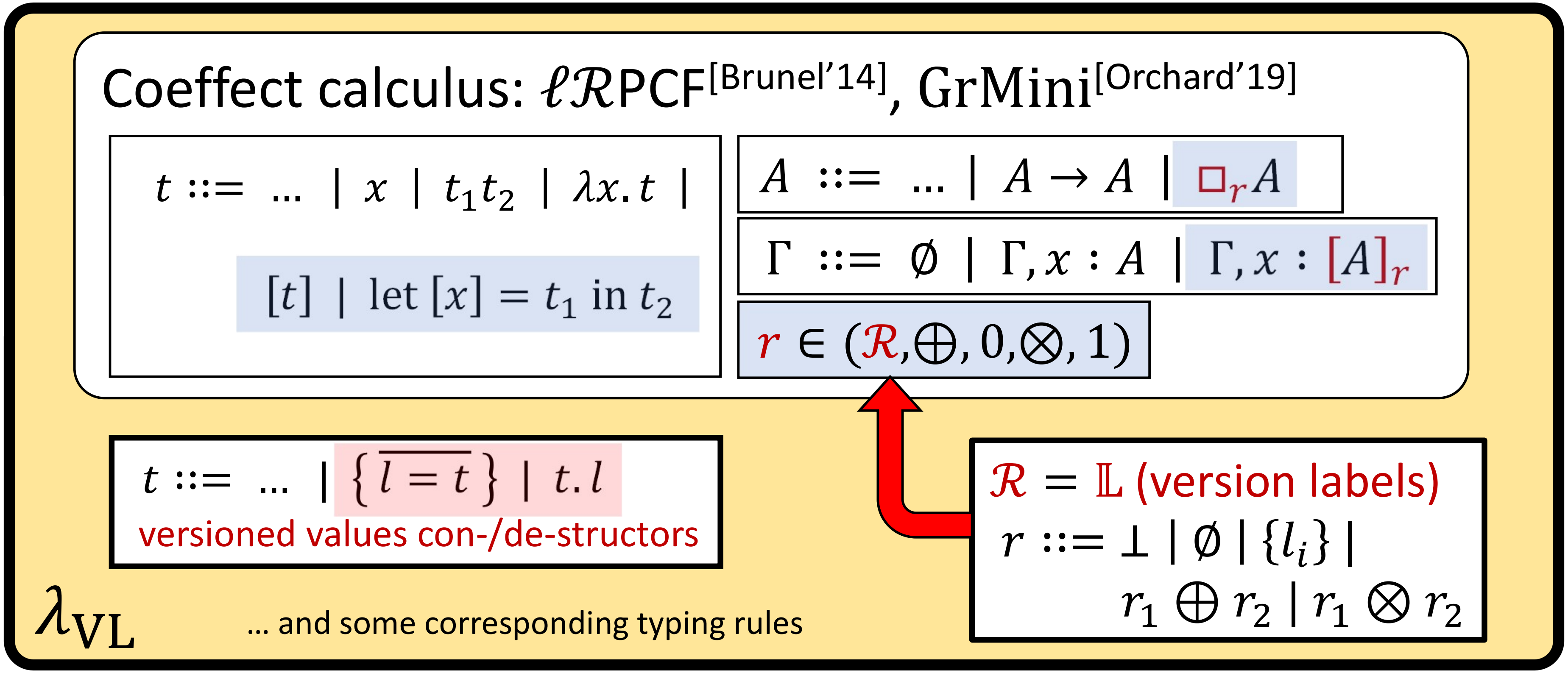
- *Indirect dependencies* complicate updates



Only help externally

- Increasing update costs
  - Lead to version locking[Preston-Werner'13]
  - Discourage users from updates[Bavota'15]

Version Resource Semiring $\mathcal{R}$



Coeffect calculus: $\ell\mathcal{R}$PCF[Brunel'14], GrMini[Orchard'19]

$t ::= \ldots \mid x \mid t_1 t_2 \mid \lambda x.t \mid$

$[t] \mid \mathrm{let}\ [x] = t_1\ \mathrm{in}\ t_2$

$A ::= \ldots \mid A \to A \mid \Box_r A$

$\Gamma ::= \emptyset \mid \Gamma, x : A \mid \Gamma, x : [A]_r$

$r \in (\mathcal{R}, \oplus, 0, \otimes, 1)$

$t ::= \ldots \mid \{\overline{l = t}\} \mid t.l$

versioned values con-/de-structors

$\mathcal{R} = \mathbb{L}$ (version labels)

$r ::= \bot \mid \emptyset \mid \{l_i\} \mid r_1 \oplus r_2 \mid r_1 \otimes r_2$

λ_{VL}

… and some corresponding typing rules

$\mathcal{R} = \{\mathrm{Irrelevant}, \mathrm{Private}, \mathrm{Public}\}$

(security level[Orchard'19])

e.g. $\Box_{\mathrm{Private}} A, \Box_{\mathrm{Public}} A$

$\mathcal{R} = \mathbb{N}$ (exact usage[Petricek'14])

e.g. $\Box_0 A, \Box_2 A$

Version Awareness

Additive part: *resource splitting*

$$\frac{\underline{\Gamma_1} \vdash t_1 : A \to B \quad \underline{\Gamma_2} \vdash t_2 : A}{\underline{\Gamma_1 + \Gamma_2} \vdash t_1\ t_2 : B}\ \text{app}$$

Splitting resources for sub judgments

$$(\Gamma, x : [A]_r) + (\Gamma', x : [A]_s) = (\Gamma + \Gamma'), x : [A]_{r \oplus s}$$

Multiplication part: *resource demanding*

$$\frac{[\Gamma] \vdash t : A}{\underline{r * [\Gamma]} \vdash [t] : \underline{\Box_r A}}\ \text{pr}$$

Requiring resources from a context

$$r * (\Gamma, x : [A]_s) = (r \cdot \Gamma), x : [A]_{r \otimes s}$$

“$[t]$ available in r requires
all assumptions to be available in r.”

Background – λ_{VL} Type System

Intuition to 0 and 1 in Semiring

Both 0 and 1 indicate unavailable resources.

Treated differently only in multiplication $\otimes$.

$$r_1 \otimes r_2 = \begin{cases} \bot & (r_1 = \bot \vee r_2 = \bot) \\ \mathrm{r}_1 \cup r_2 & (otherwise) \end{cases}$$

$$\frac{\Gamma \vdash t : A}{\Gamma, [\Delta]_{\underset{=\bot}{0}} \vdash t : A}\,\text{weak} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_{\underset{=\emptyset}{1}} \vdash t : B}\,\text{der}$$

$$\frac{\Gamma, x : [A]_r, \Gamma' \vdash t : B \quad r \sqsubseteq s}{\Gamma, x : [A]_s, \Gamma' \vdash t : B}\,\text{sub} \quad \bot \sqsubseteq \emptyset \sqsubseteq \{l_i\} \sqsubseteq \cdots$$

In other coeffect calculi, the semantic difference between 0 and 1 may be meaningful.

i.e.) Exact usage $(\mathbb{N}, +, 0, \cdot, 1, \equiv)$[Patriceik’14,Orchard’19]

λ_{VL} Typing Rules

$$\frac{}{\emptyset \vdash n : \mathrm{Int}}\;\text{int} \qquad \frac{}{x : A \vdash x : A}\;\text{var}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \to B}\;\text{abs}$$

$$\frac{\Gamma_1 \vdash t_1 : A \to B \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1 + \Gamma_2 \vdash t_1\, t_2 : B}\;\text{app} \qquad \frac{\Gamma_1 \vdash t_1 : \Box_r A \qquad \Gamma_2, x : [A]_r \vdash t_2 : B}{\Gamma_1 + \Gamma_2 \vdash \mathbf{let}\,[x] = t_1\;\mathbf{in}\;t_2 : B}\;\text{let}$$

$$\frac{\Gamma \vdash t : A}{\Gamma, [\Delta]_0 \vdash t : A}\;\text{weak} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_1 \vdash t : B}\;\text{der} \qquad \frac{[\Gamma] \vdash t : A}{r * [\Gamma] \vdash [t] : \Box_r A}\;\text{pr}$$

$$\frac{\Gamma, x : [A]_r, \Gamma' \vdash t : B \qquad r \sqsubseteq s}{\Gamma, x : [A]_s, \Gamma' \vdash t : B}\;\text{sub} \qquad \frac{\Gamma \vdash t : \Box_r A \qquad l \in r}{\Gamma \vdash t.l : \Box_r A}\;\text{extr}$$

$$\frac{[\Gamma_i] \vdash t_i : A}{\bigcup(\{l_i\} * [\Gamma_i]) \vdash \langle \overline{l = t} \mid l_i \rangle : A}\;\text{veri} \qquad \frac{[\Gamma_i] \vdash t_i : A}{\bigcup(\{l_i\} * [\Gamma_i]) \vdash \{\overline{l = t} \mid l_i\} : \Box_{\{\bar{l}\}} A}\;\text{ver}$$

Background – λ_{VL} Type System

Properties

Well-typed versioned substitutions

(Well-typed linear substitutions hold as well)

Proved

$$\begin{cases} [\Delta] \vdash t' : A \\ \Gamma, x : [A]_r, \Gamma' \vdash t : B \end{cases} \Longrightarrow \Gamma + r \cdot \Delta + \Gamma' \vdash [t' \mapsto x]t : B$$

Type-safe extractions

Proved

$$[\Gamma] \vdash v : \Box_r A \Longrightarrow \forall \boldsymbol{l_k} \in \boldsymbol{r}. \exists t'. \begin{cases} v.\boldsymbol{l_k} \longrightarrow t' \\ [\Gamma] \vdash t' : A \end{cases}$$

①Translate **VL** to **VLMini**

Girard’s Translation

A generalization of the original for linear λ-calculus[Girard’87] to GrMini[Orchard’19]

Inserting ***syntactic annotation*** [_] where a value should be treated as a versioned value

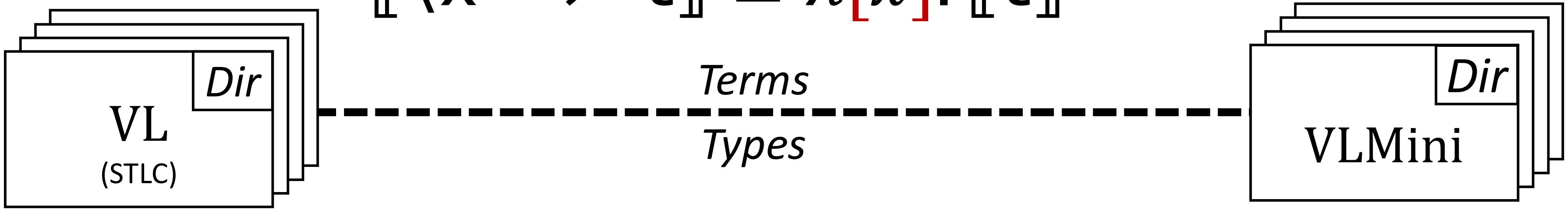
Treat arguments as versioned values …

$$[\![\texttt{x}]\!] \equiv x$$

$$[\![\texttt{t s}]\!] \equiv [\![\texttt{t}]\!]\ [[\![\texttt{s}]\!]]$$

$$[\![\texttt{\textbackslash x -> t}]\!] \equiv \lambda[x].[\![\texttt{t}]\!]$$

… and capture their version resources by promoted patterns

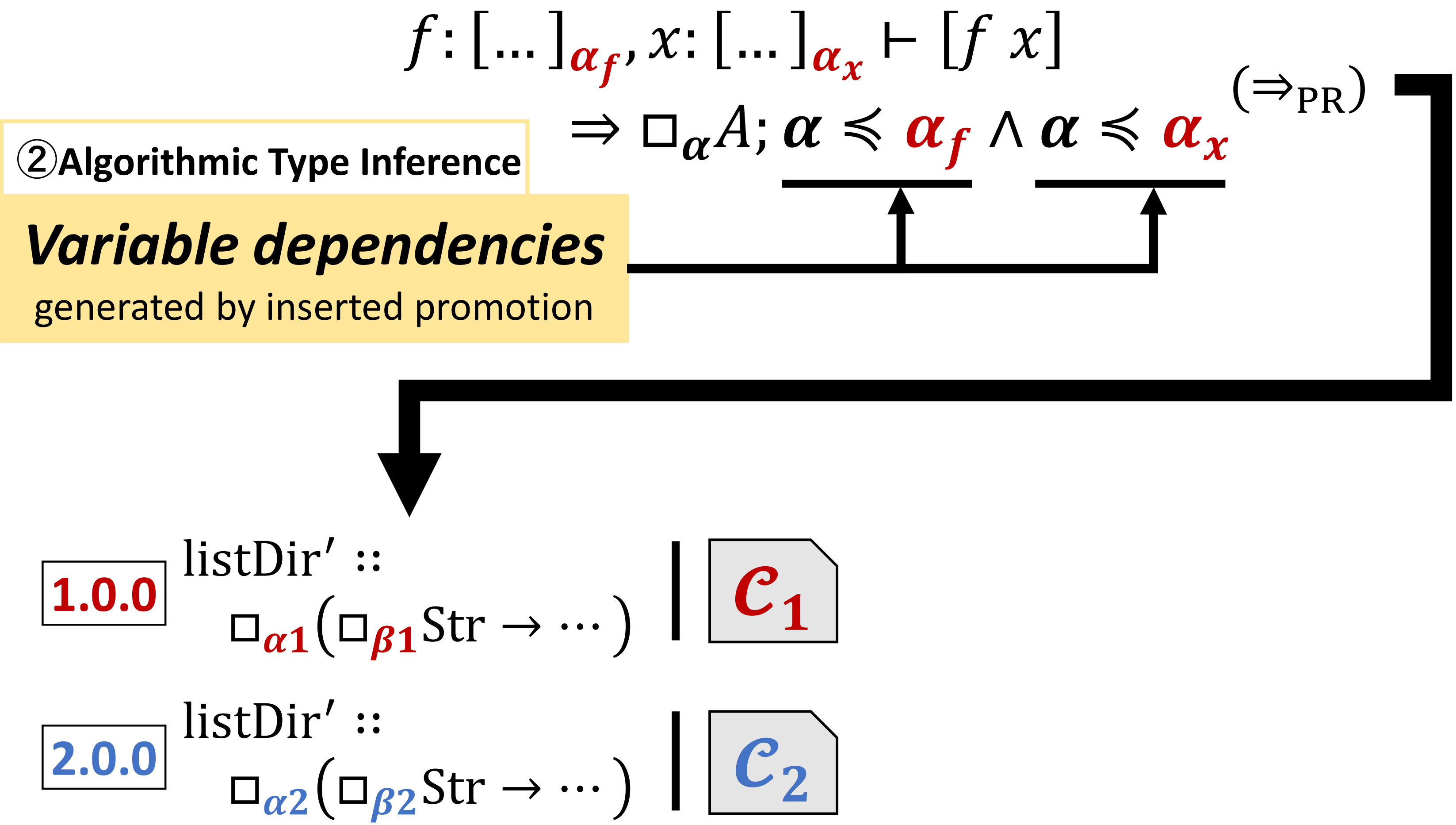


$$[\![\texttt{A}]\!] \equiv A$$

$$[\![\texttt{A -> A}]\!] \equiv \Box_r [\![A]\!] \to [\![B]\!]$$

②Version Inference

Constraint Generation



$$f : [\ldots]_{\alpha_f}, x : [\ldots]_{\alpha_x} \vdash [f\ x]$$

$$\Rightarrow \Box_{\alpha} A ; \underline{\boldsymbol{\alpha} \preccurlyeq \boldsymbol{\alpha_f}} \wedge \underline{\boldsymbol{\alpha} \preccurlyeq \boldsymbol{\alpha_x}} \quad (\Rightarrow_{\mathrm{PR}})$$

②**Algorithmic Type Inference**

Variable dependencies

generated by inserted promotion

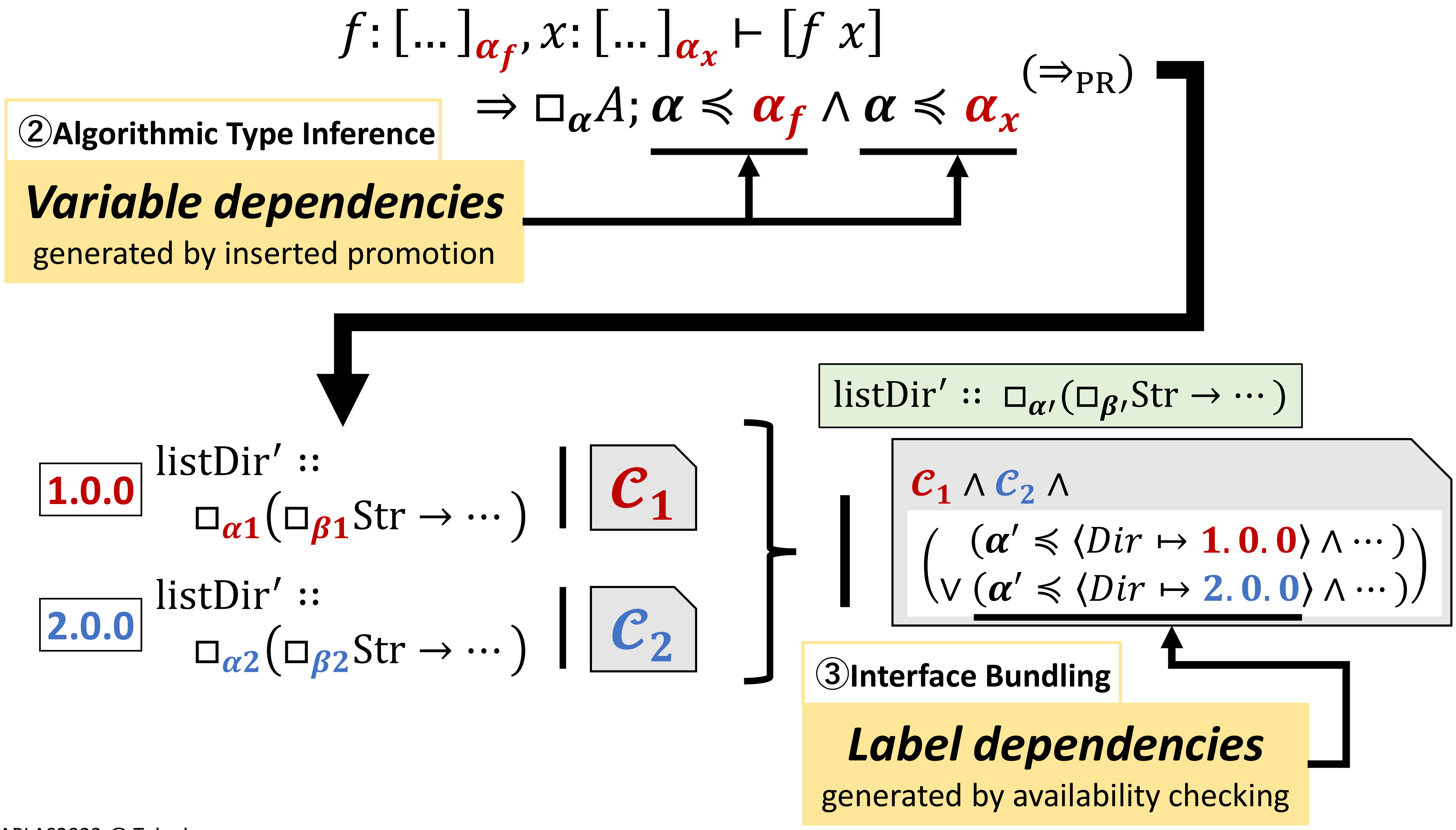
1.0.0 $\mathrm{listDir}' :: \Box_{\alpha 1}(\Box_{\beta 1}\mathrm{Str} \to \cdots) \mid \mathcal{C}_1$

2.0.0 $\mathrm{listDir}' :: \Box_{\alpha 2}(\Box_{\beta 2}\mathrm{Str} \to \cdots) \mid \mathcal{C}_2$

②Version Inference & ③Interface Bundling

Constraint Generation

Please see the paper for more details!



$f : [\dots]_{\alpha_f}, x : [\dots]_{\alpha_x} \vdash [f\ x]$

$\Rightarrow \Box_{\alpha} A; \boldsymbol{\alpha} \preccurlyeq \boldsymbol{\alpha_f} \wedge \boldsymbol{\alpha} \preccurlyeq \boldsymbol{\alpha_x}$ $(\Rightarrow_{\mathrm{PR}})$

②Algorithmic Type Inference

Variable dependencies

generated by inserted promotion

1.0.0 listDir′ :: $\Box_{\alpha 1}(\Box_{\beta 1}\mathrm{Str} \to \cdots)$ | $\mathcal{C}_1$

2.0.0 listDir′ :: $\Box_{\alpha 2}(\Box_{\beta 2}\mathrm{Str} \to \cdots)$ | $\mathcal{C}_2$

listDir′ :: $\Box_{\alpha'}(\Box_{\beta'}\mathrm{Str} \to \cdots)$

| $\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \begin{pmatrix} (\boldsymbol{\alpha'} \preccurlyeq \langle Dir \mapsto 1.0.0\rangle \wedge \cdots) \\ \vee\ (\boldsymbol{\alpha'} \preccurlyeq \langle Dir \mapsto 2.0.0\rangle \wedge \cdots) \end{pmatrix}$

③Interface Bundling

Label dependencies

generated by availability checking

② Version Inference

Algorithmic Type Inference

Allocate resource variables and *collect constraints*

$$\underbrace{\Sigma; \Gamma \vdash t}_{\textbf{Input}} \Rightarrow \underbrace{A; \Sigma'; \theta; \mathcal{C}}_{\textbf{Output}}$$

t : Term

Γ : Typing context

Σ : Type variable kinds

A : Type

$\mathcal{C}$: Constraints

Σ' : Type variable kinds

θ : Substitution

② Version Inference

Pattern Type Synthesis

$$\boxed{(\lambda[x].t)\ [y]}$$

Resource contexts

$$\underbrace{\Sigma; R \vdash p : A}_{\textbf{Input}} \rhd \underbrace{\Gamma; \Sigma'; \theta}_{\textbf{Output}}$$

Aggregate resources by $[p]$

$$\frac{\Sigma'; \alpha \vdash p : \beta \rhd \Delta; \Sigma''; \theta \quad \Sigma' \vdash A \sim \Box_{\alpha}\beta \rhd \theta'}{\Sigma; - \vdash [p] : A \rhd \Delta; \Sigma''; \theta \uplus \theta'}\ (\mathrm{p}\Box)$$

Convert the resource into assumption

$$\frac{\Sigma \vdash A : \mathrm{Type} \quad \Sigma \vdash r : \mathrm{Labels}}{\Sigma; r \vdash x : A \rhd x : [A]_{r}; \Sigma; \emptyset}\ [\mathrm{pVar}]$$

② Version Inference

Data Structure Support

- Inserting *distributive combinators*[Huges’21]

LC

$\mathbf{let}\ p = (x, y)\ \mathbf{in}$
$\mathbf{case}\ p\ \mathbf{of}\ (x, y) \mapsto \cdots$

➡

λ_{VL}

$\mathbf{let}\ [p] = \textcolor{red}{\mathit{pull}}\ ([x], [y])\ \mathbf{in}$
$\mathbf{case}\ [p]\ \mathbf{of}\ [(x, y)] \mapsto$
$\quad \mathbf{let}\ ([x], [y]) = \textcolor{red}{\mathit{push}}\ [(x, y)]\ \mathbf{in} \ldots$

Granule
[Orchard’19, Huges’21]

```
push : (a, b)[r] -> (a[r], b[r])
push [(x, y)]  = ([x], [y])
pull : (a[n], b[m]) -> (a, b)[n ⊓ m]
pull ([x], [y]) = [(x, y)]
```

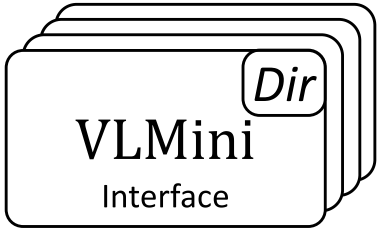
- Motivation:

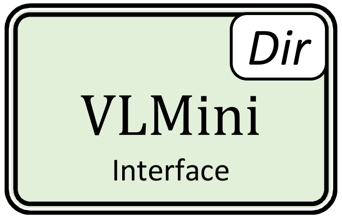
How to propagate resources in-/out-side a data structure?

	A versioned value of a tuple	*A tuple of versioned values*
$fst\ p$ is ill-typed	$fst :: \square_{r'}(\mathrm{Int}, \mathrm{Int}) \to \mathrm{Int}$ $fst = \lambda[x].\ \mathbf{case}\ [x]\ \mathbf{of}$ $[(x, y)] \mapsto x$	$p :: (\square_{r}\mathrm{Int}, \square_{s}\mathrm{Int})$ $p = ([x], [y])$

③ Interface Bundling

Generate Multi-version Interface





listDir′ :: $\square_{\alpha'}(\square_{\beta'}\text{String} \to \cdots)$

1.0.0

listDir′ :: $\square_{\alpha 1}(\square_{\beta 1}\text{String} \to \cdots)$ $\mid \mathcal{C}_1$

2.0.0

listDir′ :: $\square_{\alpha 2}(\square_{\beta 2}\text{String} \to \cdots)$ $\mid \mathcal{C}_2$

Restriction
All versions must have a same type

Generated constraints

$$\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \left(\begin{pmatrix} \alpha' \preccurlyeq \langle Dir \mapsto 1.0.0 \rangle \\ \wedge\ \alpha' \preccurlyeq \alpha_1 \\ \wedge\ \beta' \preccurlyeq \beta_1 \end{pmatrix} \vee \begin{pmatrix} \alpha' \preccurlyeq \langle Dir \mapsto 2.0.0 \rangle \\ \wedge\ \alpha' \preccurlyeq \alpha_2 \\ \wedge\ \beta' \preccurlyeq \beta_2 \end{pmatrix} \right)$$

for 1.0.0

for 2.0.0

$$\approx \square_{\left\{\begin{matrix}[Dir \mapsto 1.0.0], \\ [Dir \mapsto 2.0.0]\end{matrix}\right\}}(\square_r \text{Int} \to \text{Int})$$

$\boxed{\lambda_{\text{VL}}}$

[SBV] : http://leventerkok.github.io/sbv/

Implementation – How to use SMT solver

Vectorizing Constraints

Translate constraints to *symbolic lists*[SBV]

Label / Constraints

Symbolic lists

$$\begin{bmatrix} A \mapsto 1.0.0 \\ B \mapsto 2.0.0 \end{bmatrix} \approx [1_A, 2_B]$$

$$\alpha_2 \preccurlyeq \langle B \mapsto 2.0.0 \rangle \approx v_{\alpha 2}.2 = 2_B$$

$$\alpha_1 \preccurlyeq \alpha_2 \approx \forall i.\ v_{\alpha 1}.i = v_{\alpha 2}.i$$

M_i	A	B
id_{mod}	1	2

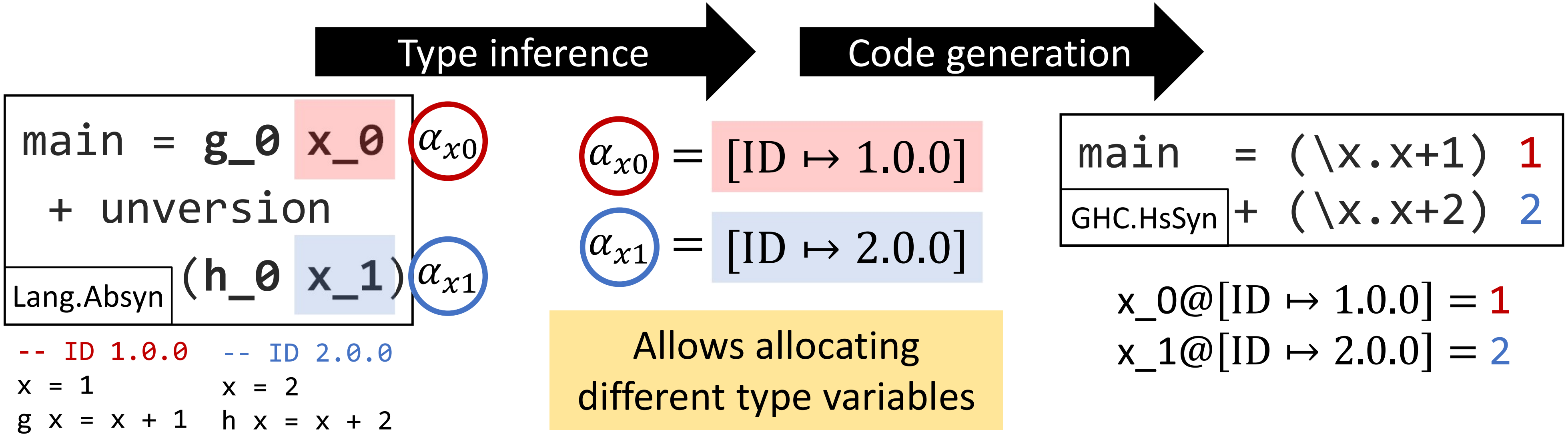
id_{ver}	A	B
1.0.0	1_A	1_B
2.0.0	2_A	2_B

A label $[M_i \mapsto V_i]$ indicates that
the $id_{mod}(M_i)$-th element of a symbolic list is $id_{ver}(M_i, V_i)$.

Implementation – Version Polymorphism

Ad-hoc Polymorphism *via* Duplication

- ***Rename*** all occurrences of ***external symbols***
  - Replicate those in constraints and contexts as well



Type inference

Code generation

Lang.Absyn

```
main = g_0 x_0
  + unversion
    (h_0 x_1)
```

α_{x0} α_{x1}

```
-- ID 1.0.0       -- ID 2.0.0
x = 1             x = 2
g x = x + 1       h x = x + 2
```

$\alpha_{x0} = [\mathrm{ID} \mapsto 1.0.0]$

$\alpha_{x1} = [\mathrm{ID} \mapsto 2.0.0]$

Allows allocating different type variables

GHC.HsSyn

```
main  = (\x.x+1) 1
      + (\x.x+2) 2
```

$\mathrm{x\_0@}[\mathrm{ID} \mapsto 1.0.0] = 1$

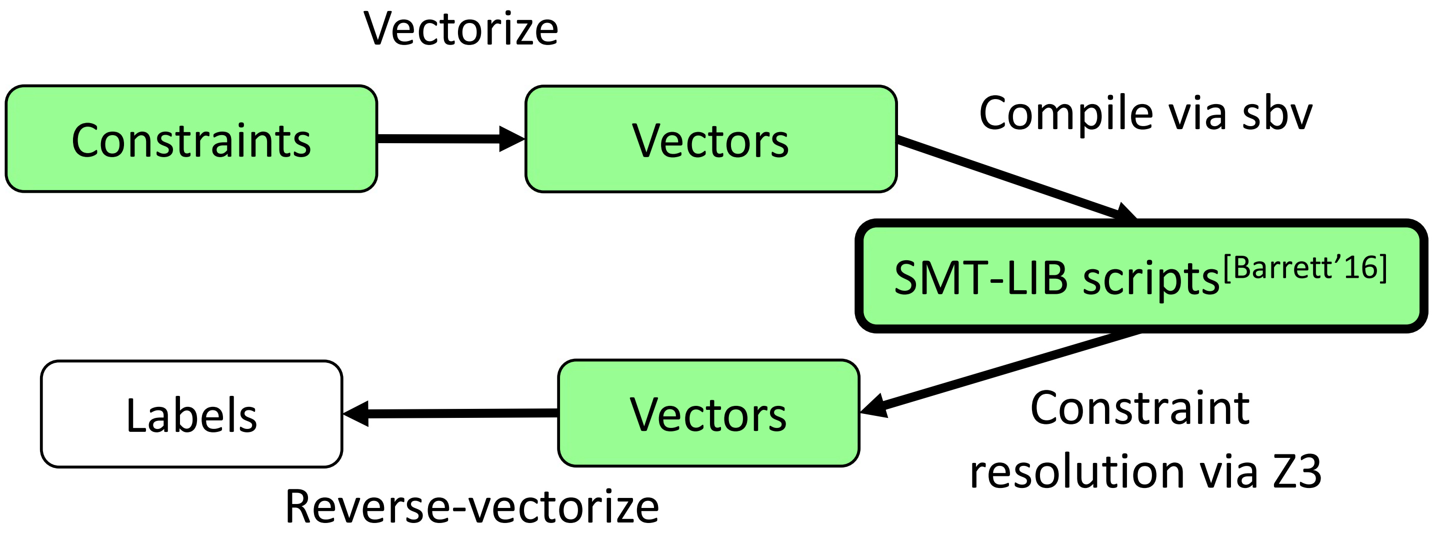
$\mathrm{x\_1@}[\mathrm{ID} \mapsto 2.0.0] = 2$

※ Full-resource polymorphism[Orchard'19] requires a revised compilation scheme and an extension to core calculus.

Compiler Performance

How to Estimate Complexity

Exponential for ***number of variables***, how to estimate?



Vectorize

Constraints

Vectors

Compile via sbv

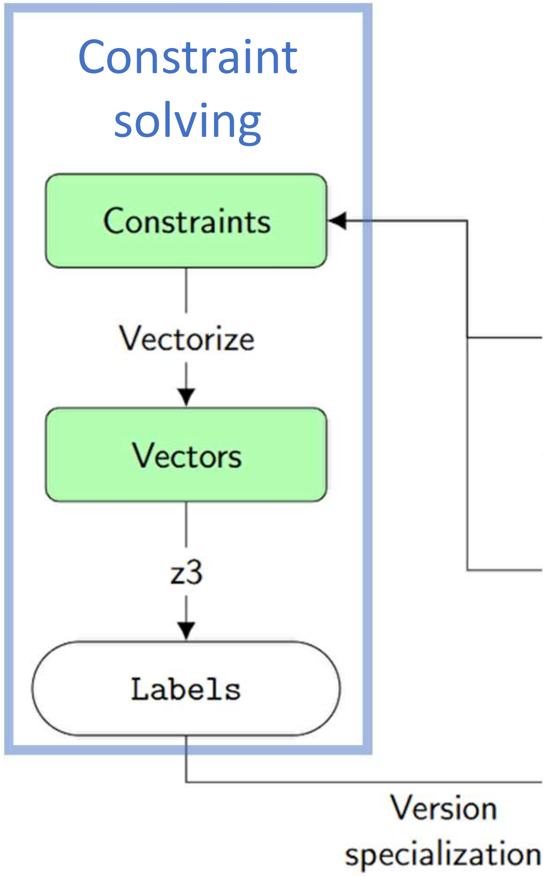
SMT-LIB scripts[Barrett'16]

Constraint resolution via Z3

Vectors

Labels

Reverse-vectorize



Constraint solving

```
(declare-const a Bool)
(declare-const b Bool)
(define-fun demorgan () Bool
     (= (and a b) (not (or (not a) (not b)))))
(assert (not demorgan))
(check-sat)
```

SMT-Lib Scripts

```
(declare-fun s0 () (_ BitVec 8))
(declare-fun s1 () (_ BitVec 8))
…
(define-fun s1552 () (_ BitVec 8) #x00)
```

Declare symbolic variables

```
                 -- Special int value indicating undefined version
…
(define-fun s1553 () Bool (distinct s1 s1552))
(define-fun s1554 () Bool (= s0 s1))
(define-fun s1555 () Bool (and s1553 s1554))
(define-fun s1556 () Bool (= s1 s1552))
(define-fun s1557 () Bool (xor s1555 s1556))
…
```

Constraints

New symbolic variables per variable/label dependency

```
(assert s3842)   -- Represents all constraints
(minimize s4362) -- Maximize the number of
                 -- undefined version elements
```

Assertion

```
(check-sat)
(get-objectives)
…
```

Inspecting solution models

Compiler Performance

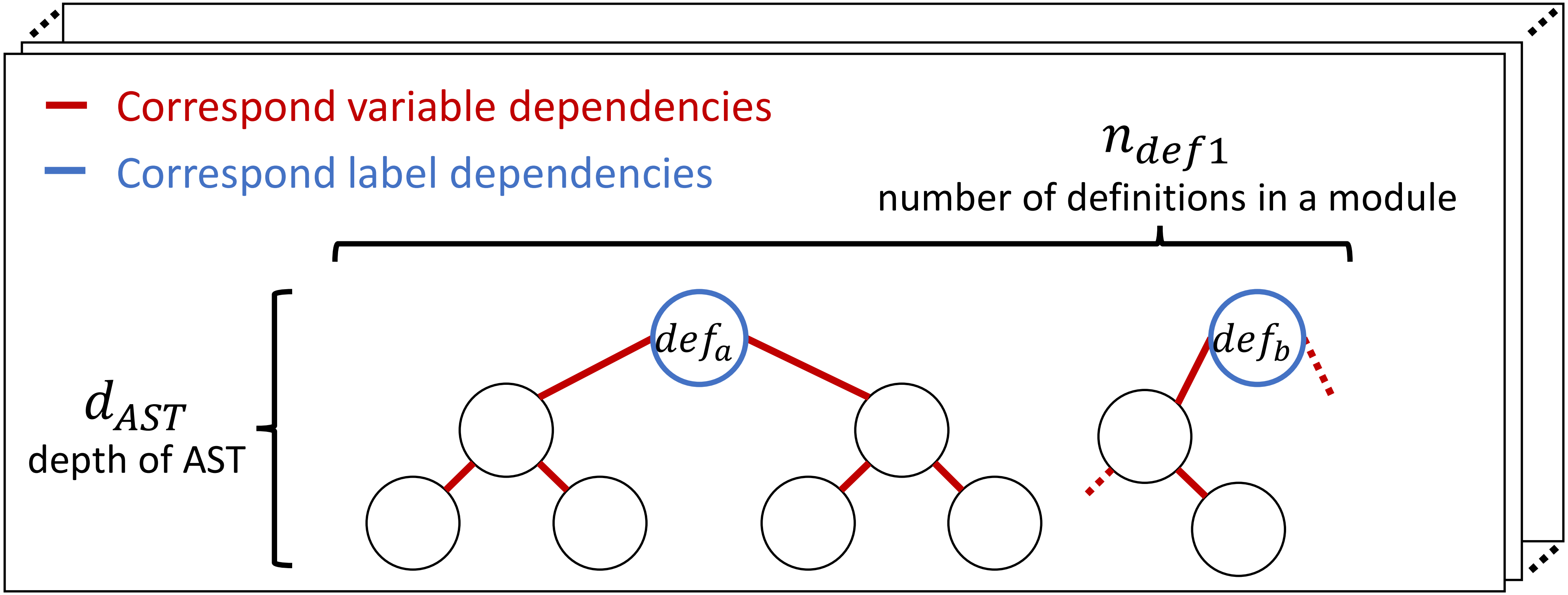
Size of Constraints

Size of *variable dep.* : $O\left(\overbrace{n_{mod}}^{\text{Vector size}}\overbrace{n_{def}}^{\text{Number of definitions}}\overbrace{\mathbf{2}^{d_{AST}}}^{\text{Number of AST edges}}\right)$

Size of *label dep.* : $O(n_{mod}n_{def})$

$O(n_{def}) = O(n_{def1}n_{ver}n_{mod})$

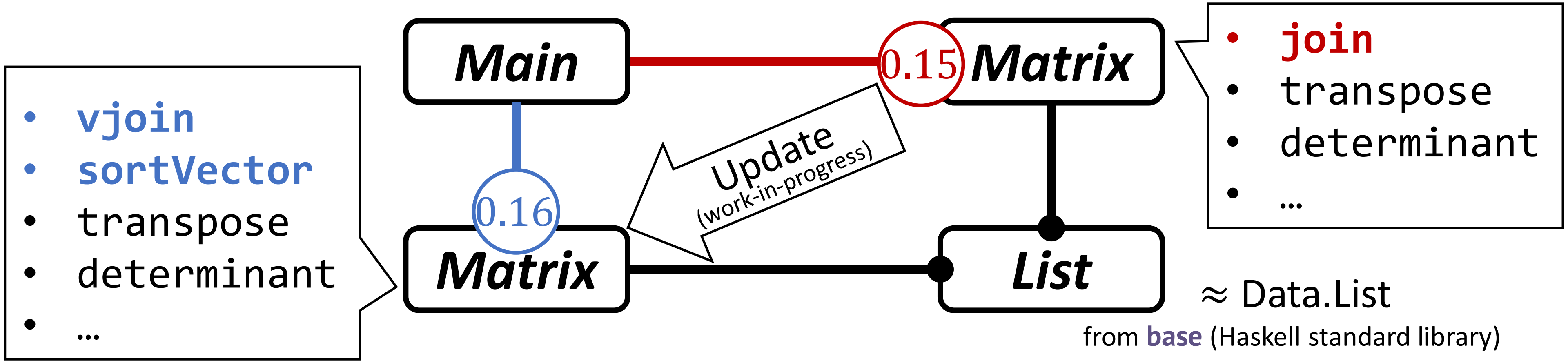
$\times\ n_{ver} \times n_{mod}$


Correspond variable dependencies
Correspond label dependencies
n_{def1}
number of definitions in a module
d_{AST}
depth of AST
def_a
def_b

Case Study – **hmatrix**

Module Structure

Reproducing incompatibility with ***Matrix*** and ***List***



- `vjoin`
- `sortVector`
- `transpose`
- `determinant`
- …

- `join`
- `transpose`
- `determinant`
- …

≈ Data.List
from **base** (Haskell standard library)

version	`join`	`sortVector`
< 0.15.0	***available***	unavailable
≥ 0.16.0	Deleted (replaced `vjoin`)	***available***

hmatrix

```
Vector a  =>  List Int
Matrix a  =>  List (List Int)
```

Matrix

Case Study – **hmatrix**

Handling Multiple Versions in a Code

Compile

Dispatching to a consistent version of programs, with **unversion** as boundaries

```
main =
  let vec    = [2, 1]
      sorted = unversion
                   (sortVector vec)
      m22 = join -- [[1,2],[2,1]]
                (singleton sorted)
                (singleton vec)
  in determinant m22
```

Haskell AST (prettyprint)

```
module Main where
main = ...
    -- join
    (let join = \xs -> \ys ->
        case xs of
            [] -> ys
            x : xs -> (:) x (join xs ys)
    in join)
  ...
    -- sortVector
    (let sortVector = \xs ->
        case xs of
            []  -> []
            [x] -> [x]
            xs  -> (\r -> (let vjoin = ...  in vjoin)
                                (sortVector
                                  ((let init = ... in init) r))
                                [(let last = ... in last) r])
                      ((let bubble = ... in bubble) xs)
    in sortVector)
  ...
```

Consistent in 0.15

Consistent in 0.16

```
> ghc -o main Main.hs
> ./main
-3
```

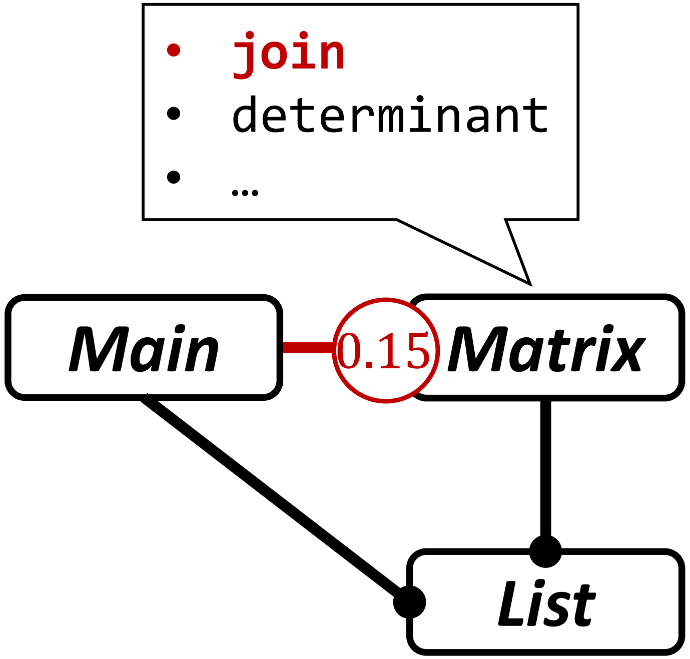
Note: inserting IO function “print” manually

Case Study – **hmatrix**

Main Module (*before* update)

Haskell

```
module Main where

import Matrix
import List

main =
  let vec  = [2, 1]
      vec’ = [1, 2]
      m22 = join -- [[1,2],[2,1]]
              (singleton vec’)
              (singleton vec)
  in determinant m22
```



- join
- determinant
- …

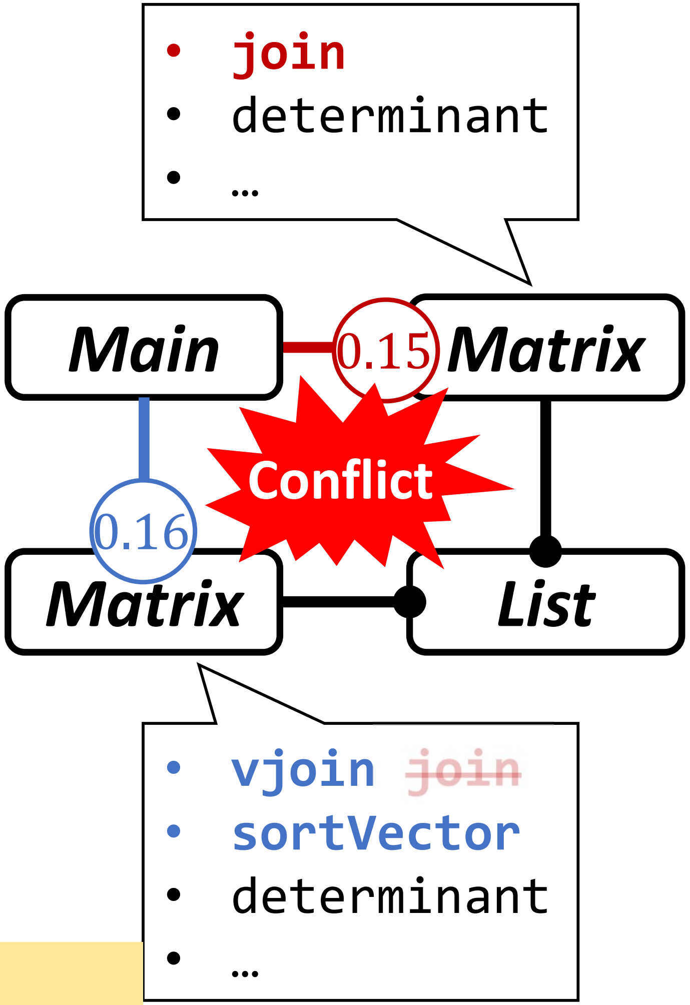
Case Study – **hmatrix**

Main Module (*after* update)

Haskell

```haskell
module Main where

import Matrix
import List

main =
  let vec   = [2, 1]
      sorted = sortVector vec
      m22 = join -- [[1,2],[2,1]]
              (singleton sorted)
              (singleton vec)
  in determinant m22
```

```
main.hs:1:38: error:
    Variable not in scope: sortVector
```



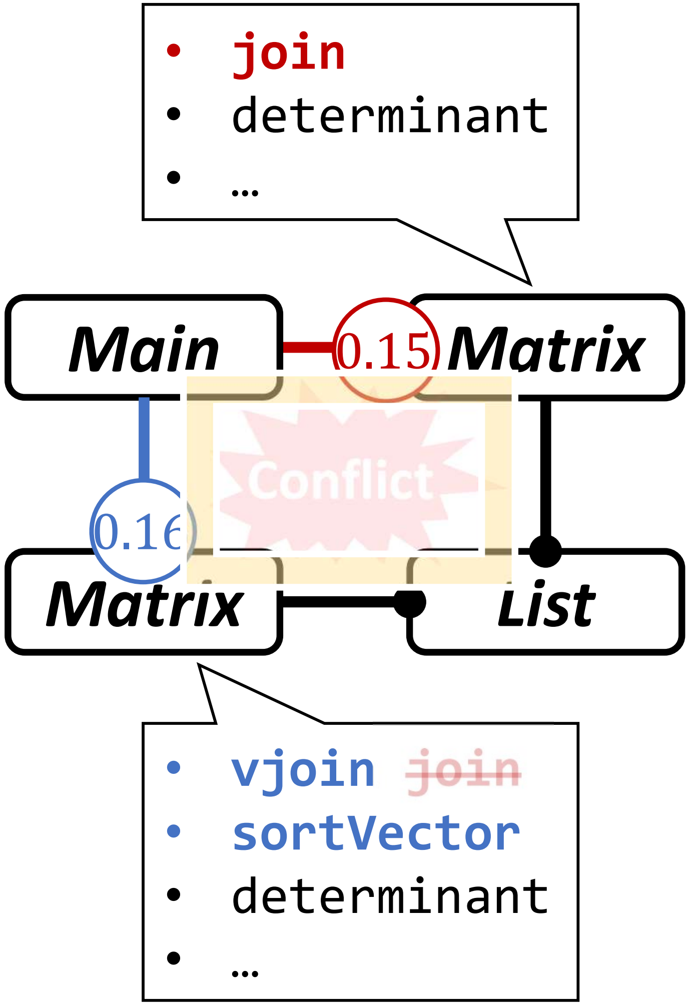
Case Study – **hmatrix**

Detecting Inconsistent Version

```
module Main where                    VL

import Matrix
import List

main =
  let vec    = [2, 1]
      sorted = sortVector vec
      m22 = join -- [[1,2],[2,1]]
          (singleton sorted)
          (singleton vec)
  in determinant m22
```

Inconsistent

Version conflicts resolved,
but ***no consistent versions***



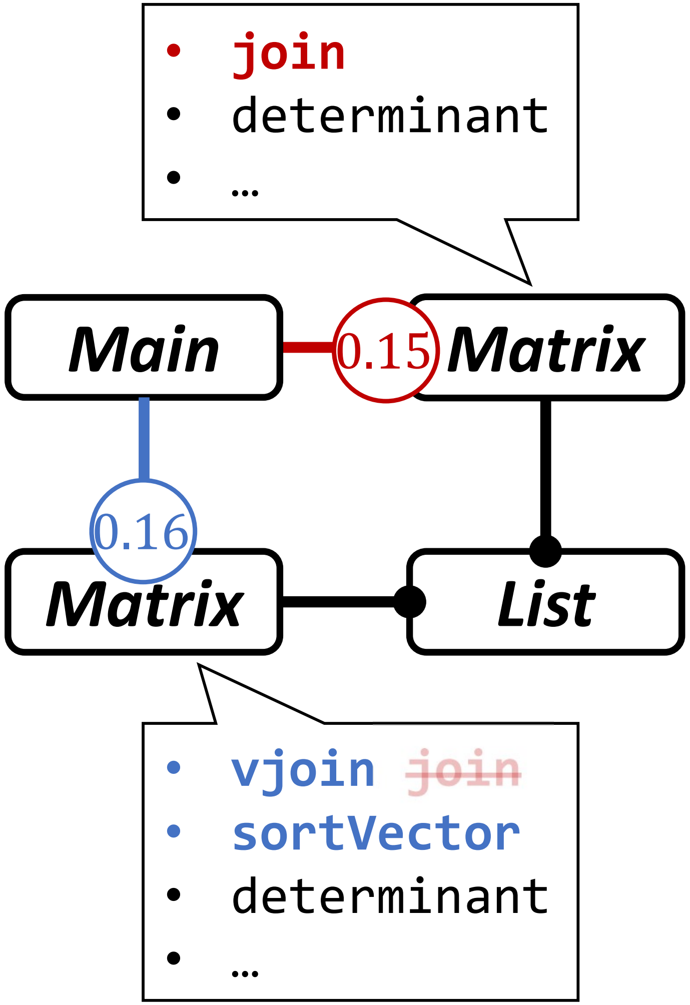
Case Study – **hmatrix**

Handling Two Versions in One Client

VL

```
module Main where

import Matrix
import List

main =
  let vec    = [2, 1]
      sorted = unversion
                  (sortVector) vec
      m22 = join -- [[1,2],[2,1]]
                (singleton sorted)
                (singleton vec)
  in determinant m22
```

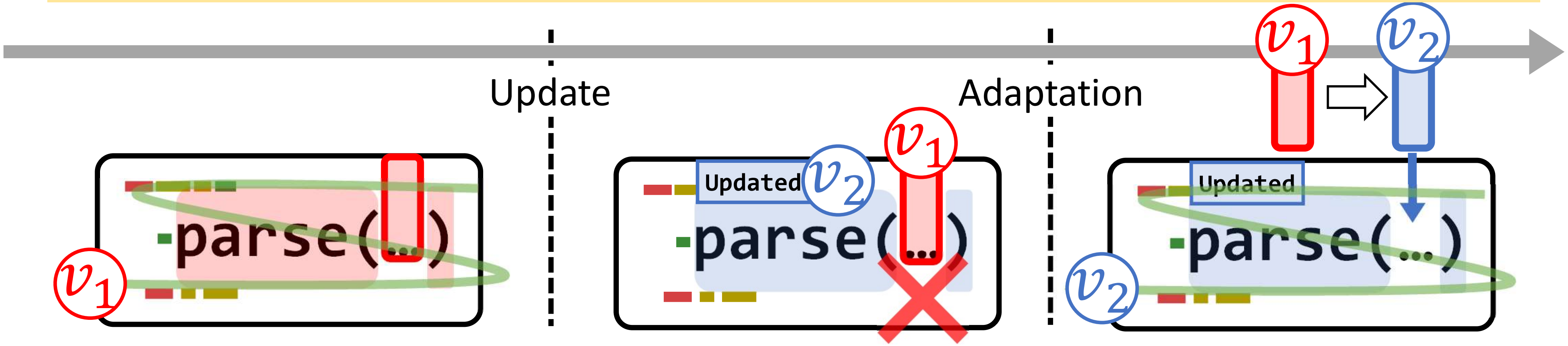
No longer depends on 0.16



Future Work

Automatic Adaptations

Detecting incompatibilities and ***inserting adapters*** automatically



Concept:

- **Code repository**, a *persistent definition/package store*
- Working environment(s) that are ***views*** into the code repository

Nix [Dolstra'04]: Hash-tagged packages + Nix package manager
Unison: Hash-tagged definitions + Unison code base manager

+ ***Consistency checking within expressions***