

Filling Typed Holes with Live GUIs

STRUMENTA

Cyrus Omar, David Moon, Andrew Blinn

Future of Programming Lab (FP Lab) @ University of Michigan

Ian Voysey

Carnegie Mellon University

Nick Collins, Ravi Chugh

University of Chicago

Creative End Users



Programmers

A screenshot of a code editor (VS Code) showing a file named `browser-menu-plugin.ts`. The code is written in TypeScript and defines a class `BrowserMainMenuFactory` that implements a `MenuModelRegistry` interface. The class has a protected readonly property `menuProviders`. It includes methods for creating a dynamic menubar and adding items to it. The code editor also shows several error messages in the Problems panel, such as "Property 'act' does not exist on type 'DynamicMenuBarWidget'", "unused expression, expected an assignment or function call (no-unused-expression)", and "Missing semicolon (semicolon) [10000]".

```
class BrowserMainMenuFactory implements MenuModelRegistry {
    private menuProviders: Map<string, MenuProvider> = new Map();
    private logger: Logger;
    private contextKeyService: ContextKeyService;
    private constructor: Constructor;
    private commandRegistry: CommandRegistry;
    private keybindingRegistry: KeybindingRegistry;
    private menuProvider: MenuProvider;

    constructor(logger: Logger, contextKeyService: ContextKeyService, constructor: Constructor, commandRegistry: CommandRegistry, keybindingRegistry: KeybindingRegistry) {
        this.logger = logger;
        this.contextKeyService = contextKeyService;
        this.constructor = constructor;
        this.commandRegistry = commandRegistry;
        this.keybindingRegistry = keybindingRegistry;
        this.menuProvider = new DynamicMenuBarWidget();
    }

    register(menuProvider: MenuProvider): void {
        this.menuProviders.set(menuProvider.id, menuProvider);
    }

    getMenuModel(): MenuModel {
        return this.menuProvider.getMenuModel();
    }

    getMenuProviders(): MenuProvider[] {
        return [...this.menuProviders.values()];
    }

    createDynamicMenuBar(): DynamicMenuBarWidget {
        const menuBar = new DynamicMenuBarWidget();
        menuBar.id = 'theimenubar';
        const menuModel = this.menuProvider.getMenu(MAIN_MENU_BAR);
        const phosphorCommands = this.createPhosphorCommands(menuModel);
        if (!phosphorCommands) {
            this.logger.error('No commands found to be visible');
        }
        phosphorCommands.invisible = () => true;
        menuBar.act(phosphorCommands);
        return menuBar;
    }

    createDynamicContextMenuItem(path: string, anchor?: Anchor): MenuItem {
        const menuModel = this.menuProvider.getMenu(path);
        const menuWidget = new DynamicMenuItemWidget(menuModel, anchor);
        menuWidget.act(phosphorCommands);
        return menuWidget;
    }

    createContextMenu(path: string, anchor?: Anchor): MenuWidget {
        const menuModel = this.menuProvider.getMenu(path);
        const phosphorCommands = this.createPhosphorCommands(menuModel, anchor);
        const contextMenuItem = new DynamicMenuItemWidget(menuModel, { commands: phosphorCommands }, this.contextKeyService);
        contextMenuItem.act(phosphorCommands);
        return contextMenuItem;
    }

    updateSubMenus(): void {
        this.menuProvider.updateSubMenus();
    }

    openSubMenus(): void {
        this.menuProvider.openSubMenus();
    }

    closeSubMenus(): void {
        this.menuProvider.closeSubMenus();
    }

    onStart(): void {
        this.menuProvider.onStart();
    }

    createLogo(): void {
        this.menuProvider.createLogo();
    }
}
```

Live & Direct



Abstract & Symbolic

A screenshot of a code editor (VS Code) showing a file named 'browser-menu-plugin.ts'. The code is written in TypeScript and defines a class 'BrowserMainMenuFactory' that implements a 'MenuFactory' interface. The class has a constructor that takes a 'menuProvider' parameter. It contains methods for creating a dynamic menu bar and a context menu. The code uses the 'phosphorCommands' and 'phosphorCommands' interfaces from the 'phosphor' library. The 'createMenuBar' method creates a 'DynamicMenuBarWidget' with a specific ID and adds it to the menu provider. The 'createContextMenu' method creates a 'DynamicMenuItemWidget' with a specific path and adds it to the menu provider. The code editor also shows a 'Problems' panel with several errors related to the 'act' method and semicolons. The 'OUTLINE' panel on the right shows the structure of the project, including files like 'BrowserMainMenuFactory', 'logger', 'contextKeyService', 'constructor', 'commandRegistry', 'keybindingRegistry', 'menuProvider', 'createMenuBar', 'createContextMenuItem', 'DynamicMenuBarWidget', 'DynamicMenuItemWidget', 'openSubMenus', 'updateSubMenus', and 'BrowserMenuBarContrib...'.

Live & Direct



Abstract & Symbolic

A screenshot of a code editor (VS Code) showing a file named "browser-menu-plugin.ts". The code is written in TypeScript and defines a class "BrowserMainMenuFactory" with a static method "createMenuBar". The method creates a "DynamicMenuBarWidget" with an ID of "thimbar", gets a "menuProvider" from "menuModel", and adds commands to it. It then activates the menu bar. The code editor also shows a sidebar with project files and an outline of the code structure. There are several error messages in the bottom left corner related to the "act" method and semicolons.

- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

Live & Direct



- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

Abstract & Symbolic

A screenshot of a code editor, likely VS Code, showing a file named 'browser-menu-plugin.ts'. The code is written in TypeScript and defines a class 'BrowserMainMenuFactory' with methods for creating menu bars and context menus. The 'Problems' panel at the bottom shows several errors, including 'Property 'act' does not exist on type 'DynamicMenuBarWidget'', 'Unresolved identifier 'phosphorCommands'' (with a red squiggle under it), and 'Missing semicolon'. The 'Outline' panel on the right shows the structure of the project, including files like 'THEIA', 'node_modules', 'src', and 'browser-menu-plugin.ts'.

- + Generic symbolic representations
- + Symbol manipulation affordances
- + Abstraction and composition and calculation and automation

Live & Direct & *Abstract & Symbolic*

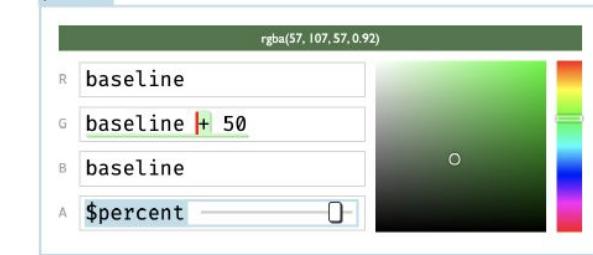


- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback
- + Generic symbolic representations
- + Symbol manipulation affordances
- + Abstraction and composition and calculation and automation

Live & Direct

Livelits (a.k.a. *live literals*) in Hazel

```
let baseline = $slider 0 255 in
let $percent = $slider 0 100 in
let default_color =
  $color
```



```
let q1_max = 36. in
let grades =
  $dataframe
```

A screenshot of the Hazel application interface showing a data frame editor. The code above defines a variable q1_max with a value of 36. and a variable grades which is a \$dataframe. The data frame has columns labeled "A1", "A2", "A3", "Midterm", and "Final". There are three rows with student names: Andrew, Cyrus, and David. The "A1" column for Andrew has a value of 80. with a yellow highlight. The "A2" column for Andrew has a value of 92. The "A3" column for Andrew has a value of 83.5. The "Midterm" column for Andrew has a value of 95. The "Final" column for Andrew has a value of 88. The "A1" column for Cyrus has a value of 61. The "A2" column for Cyrus has a value of 64. The "A3" column for Cyrus has a value of 98. The "Midterm" column for Cyrus has a value of 70. The "Final" column for Cyrus has a value of 85. The "A1" column for David has a value of 75. The "A2" column for David has a value of 81. The "A3" column for David has a value of 73. The "Midterm" column for David has a value of 82. The "Final" column for David has a value of 79. A plus sign (+) is located at the bottom right of the data frame.

- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

Abstract & Symbolic

- + Generic symbolic representations
- + Symbol manipulation affordances
- + Abstraction and composition and calculation and automation

Live & Direct

Abstract & Symbolic

Livelits (a.k.a. *live literals*) in Hazel

Live & Direct Demo! [hazel.org/livelits]

- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

- + Generic symbolic representations
- + Symbol manipulation affordances
- + Abstraction and composition and calculation and automation

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
type Color = (.r Int, .g Int, .b Int, .a Int)
livelit $color at Color {
    type Model = (.r SpliceRef, .g SpliceRef,
                  .b SpliceRef, .a SpliceRef)

    context { }

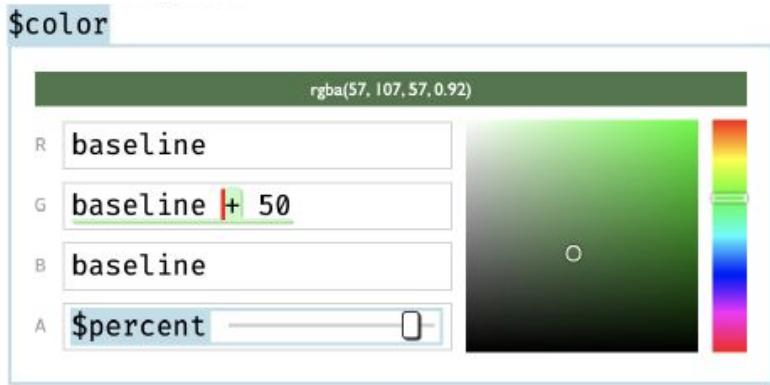
    let init : UpdateCmd(Model) = do
        r <- new_splice(`Int`, Some(`0`))
        g <- new_splice(`Int`, Some(`0`))
        b <- new_splice(`Int`, Some(`0`))
        a <- new_splice(`Int`, Some(`100`))
        return (r, g, b, a)

    ...
}
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
type Color = (.r Int, .g Int, .b Int, .a Int)
livelit $color at Color {
    type Model = (.r SpliceRef, .g SpliceRef,
                  .b SpliceRef, .a SpliceRef)

    context { }

    let init : UpdateCmd(Model) = do
        r <- new_splice(`Int`, Some(`0`))
        g <- new_splice(`Int`, Some(`0`))
        b <- new_splice(`Int`, Some(`0`))
        a <- new_splice(`Int`, Some(`100`))
        return (r, g, b, a)

    ...
}
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
type Color = (.r Int, .g Int, .b Int, .a Int)
livelit $color at Color {
    type Model = (.r SpliceRef, .g SpliceRef,
                  .b SpliceRef, .a SpliceRef)

    context { }

    let init : UpdateCmd(Model) = do
        r <- new_splice(`Int`, Some(`0`))
        g <- new_splice(`Int`, Some(`0`))
        b <- new_splice(`Int`, Some(`0`))
        a <- new_splice(`Int`, Some(`100`))
        return (r, g, b, a)
}
```

...

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
type Action =
| ClickOn(Color)

let view : Model -> ViewCmd(Html(Action)) =
fun model -> do
  (* determine a color to display *)
  r_res <- eval_splice(model.r)
  g_res <- eval_splice(model.g)
  b_res <- eval_splice(model.b)
  a_res <- eval_splice(model.a)
  let cur_color : Color =
    case (r_res, g_res, b_res, a_res)
    | (Some(Val(IntLit(r))), Some(Val(IntLit(g))), Some(Val(IntLit(b))), Some(Val(IntLit(a)))) ->
      Some((r, g, b, a))
    | _ ->
      (* indeterminate color shown as X *)
      None
  in
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
type Action =
| ClickOn(Color)

let view : Model -> ViewCmd(Html(Action)) =
fun model -> do
  (* determine a color to display *)
  r_res <- eval_splice(model.r)
  g_res <- eval_splice(model.g)
  b_res <- eval_splice(model.b)
  a_res <- eval_splice(model.a)
  let cur_color : Color =
    case (r_res, g_res, b_res, a_res)
    | (Some(Val(IntLit(r))), Some(Val(IntLit(g))), Some(Val(IntLit(b))), Some(Val(IntLit(a)))) ->
      Some((r, g, b, a))
    | _ ->
      (* indeterminate color shown as X *)
      None
  in
```

Live & Direct & *Abstract & Symbolic*

Livelit Provider API: Model-View-Update-Expand + Splice Monads



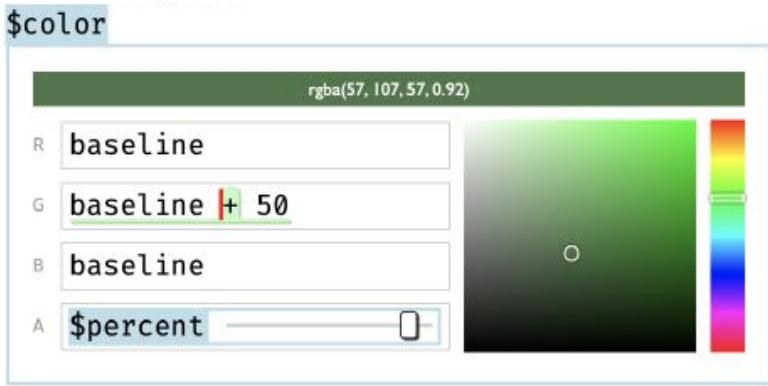
```
type Action =
| ClickOn(Color)

let view : Model -> ViewCmd(Html(Action)) =
fun model -> do
  (* determine a color to display *)
  r_res <- eval_splice(model.r)
  g_res <- eval_splice(model.g)
  b_res <- eval_splice(model.b)
  a_res <- eval_splice(model.a)
  let cur_color : Color =
    case (r_res, g_res, b_res, a_res)
    | (Some(Val(IntLit(r))), Some(Val(IntLit(g))), Some(Val(IntLit(b))), Some(Val(IntLit(a)))) ->
      Some((r, g, b, a))
    | _ ->
      (* indeterminate color shown as X *)
      None
  in
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



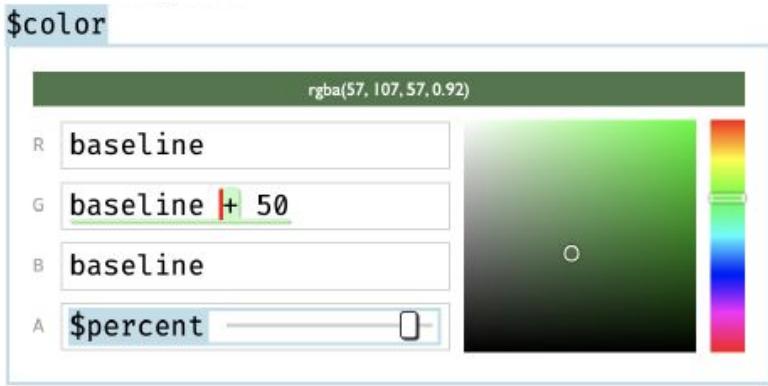
```
(* generate splice editors *)
let size = FixedWidth(20) in
r_editor <- editor(model.r, size)
g_editor <- editor(model.g, size)
b_editor <- editor(model.b, size)
a_editor <- editor(model.a, size)

(* ... now we can render the UI ... *)
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
(* generate splice editors *)
let size = FixedWidth(20) in
r_editor <- editor(model.r, size)
g_editor <- editor(model.g, size)
b_editor <- editor(model.b, size)
a_editor <- editor(model.a, size)
```

```
(* ... now we can render the UI ... *)
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
let update :  
  Model -> Action -> UpdateCmd(Model) =  
    fun model (ClickOn c) -> do  
      set_splice(model.r, IntLit(c.r))  
      set_splice(model.g, IntLit(c.g))  
      set_splice(model.b, IntLit(c.b))  
      set_splice(model.a, IntLit(c.a))  
    return model
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads

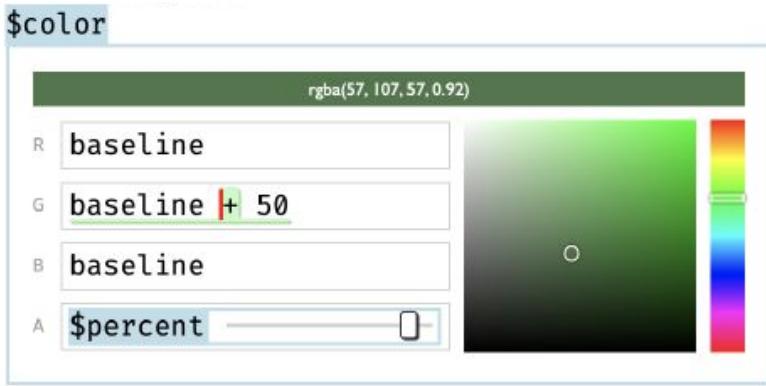


```
let update :  
    Model -> Action -> UpdateCmd(Model) =  
    fun model (ClickOn c) -> do  
        set_splice(model.r, IntLit(c.r))  
        set_splice(model.g, IntLit(c.g))  
        set_splice(model.b, IntLit(c.b))  
        set_splice(model.a, IntLit(c.a))  
    return model
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-Expand + Splice Monads



```
let expand : Model -> (Exp, List(SpliceRef)) =  
  fun model -> (^fun r g b a -> (r, g, b, a),  
    [model.r, model.g, model.b, model.a])
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-**Expand + Splice Monads**

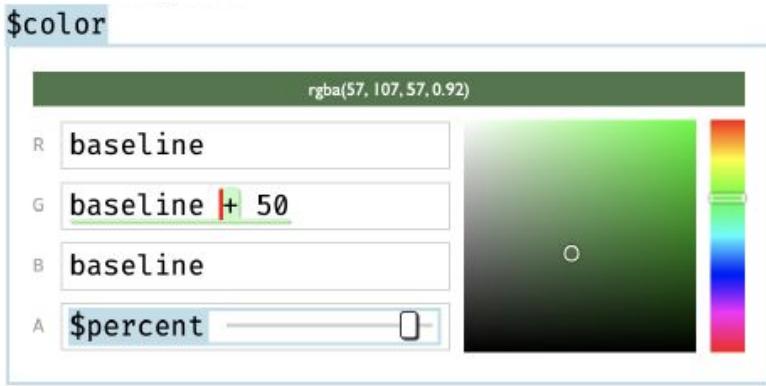


```
let expand : Model -> (Exp, List(SpliceRef)) =  
  fun model -> (^fun r g b a -> (r, g, b, a)`,  
  [model.r, model.g, model.b, model.a])
```

Live & Direct

Abstract & Symbolic

Livelit Provider API: Model-View-Update-**Expand + Splice Monads**



```
let expand : Model -> (Exp, List(SpliceRef)) =  
  fun model -> (^fun r g b a -> (r, g, b, a)`,  
    [model.r, model.g, model.b, model.a])
```

The Typed Livelit Calculus

Overview

Typ $\tau ::= \tau_1 \rightarrow \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \tau_1 + \tau_2 \mid t \mid \mu(t.\tau)$	
Unexpanded	UExp $\hat{e} ::= x \mid \lambda x. \hat{e} \mid \hat{e}_1 \hat{e}_2 \mid \dots \mid \langle \rangle^u \mid \$a \langle d_{\text{model}}; \{\psi_i\}_{i < n} \rangle^u$
External / Expanded	EExp $e ::= x \mid \lambda x. e \mid e_1 e_2 \mid \dots \mid \langle \rangle^u$
Internal	IExp $d ::= x \mid \lambda x. d \mid d_1 d_2 \mid \dots \mid \langle \rangle_\sigma^u$
Splice	$\psi ::= \hat{e} : \tau$

The Typed Livelit Calculus

Overview

	Typ	$\tau ::= \tau_1 \rightarrow \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \tau_1 + \tau_2 \mid t \mid \mu(t.\tau)$
Unexpanded	UExp	$\hat{e} ::= x \mid \lambda x. \hat{e} \mid \hat{e}_1 \hat{e}_2 \mid \dots \mid \langle \rangle^u \mid \$a \langle d_{\text{model}}; \{\psi_i\}_{i < n} \rangle^u$
External / Expanded	EExp	$e ::= x \mid \lambda x. e \mid e_1 e_2 \mid \dots \mid \langle \rangle^u$
Internal	IExp	$d ::= x \mid \lambda x. d \mid d_1 d_2 \mid \dots \mid \langle \rangle_\sigma^u$
	Splice	$\psi ::= \hat{e} : \tau$

The Typed Livelit Calculus

Overview

	Typ	$\tau ::= \tau_1 \rightarrow \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \tau_1 + \tau_2 \mid t \mid \mu(t.\tau)$
Unexpanded	UExp	$\hat{e} ::= x \mid \lambda x. \hat{e} \mid \hat{e}_1 \hat{e}_2 \mid \dots \mid \langle \rangle^u \mid \$a \langle d_{\text{model}}; \{\psi_i\}_{i < n} \rangle^u$
External / Expanded	EExp	$e ::= x \mid \lambda x. e \mid e_1 e_2 \mid \dots \mid \langle \rangle^u$
Internal	IExp	$d ::= x \mid \lambda x. d \mid d_1 d_2 \mid \dots \mid \langle \rangle_\sigma^u$
	Splice	$\psi ::= \hat{e} : \tau$



Elaboration
[Hazelnut Live,
POPL'19]

The Typed Livelit Calculus

Overview

	$\text{Typ } \tau ::= \tau_1 \rightarrow \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \tau_1 + \tau_2 \mid t \mid \mu(t.\tau)$	
Unexpanded	$\text{UExp } \hat{e} ::= x \mid \lambda x. \hat{e} \mid \hat{e}_1 \hat{e}_2 \mid \dots \mid \langle \rangle^u \mid \$a \langle d_{\text{model}}; \{\psi_i\}_{i < n} \rangle^u$	
External / Expanded	$\text{EExp } e ::= x \mid \lambda x. e \mid e_1 e_2 \mid \dots \mid \langle \rangle^u$	 Expansion
Internal	$\text{IExp } d ::= x \mid \lambda x. d \mid d_1 d_2 \mid \dots \mid \langle \rangle_\sigma^u$	
	$\text{Splice } \psi ::= \hat{e} : \tau$	

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then $\Gamma \vdash e : \tau$.*

$$\frac{\begin{array}{c} \text{ELivelit} \\ \text{livelit } \$a \text{ at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$$\frac{\begin{array}{c} \text{ELivelit} \\ \text{livelit } \$a \text{ at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$$\frac{\begin{array}{c} \text{ELivelit} \\ \text{livelit } \$a \text{ at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} \ d_{\text{model}} \Downarrow d_{\text{encoded}} \qquad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

(1) Lookup

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$$\frac{\begin{array}{c} \text{ELivelit} \\ \text{livelit } \$a \text{ at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

(2) Model Validation

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$\mathsf{ELivelit}$

$$\frac{\begin{array}{c} \text{livelit \$a at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

(3) Expansion Generation

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$$\frac{\begin{array}{c} \text{ELivelit} \\ \text{livelit } \$a \text{ at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} \ d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}} \quad (4) \text{ Decoding}$$

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$\mathsf{ELivelit}$

$$\frac{\begin{array}{c} \text{livelit \$a at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \Uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

(5) Expansion Validation

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$\mathsf{ELivelit}$

$$\begin{array}{c} \text{livelit \$a at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \boxed{\{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n}} \end{array}$$

(6) Splice Expansion

$$\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}$$

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$\mathsf{ELivelit}$

$$\frac{\begin{array}{c} \text{livelit \$a at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \Uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

(7) Conclusion

The Typed Livelit Calculus

Expansion (Mechanized in Agda)

Theorem 4.4 (Typed Expansion). *If $\Phi; \Gamma \vdash \hat{e} \rightsquigarrow e : \tau$ then
 $\Gamma \vdash e : \tau$.*

$\mathsf{ELivelit}$

$$\frac{\begin{array}{c} \text{livelit \$a at } \tau_{\text{expand}} \{ \tau_{\text{model}}; d_{\text{expand}} \} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}} d_{\text{model}} \Downarrow d_{\text{encoded}} \quad d_{\text{encoded}} \uparrow e_{\text{pexpansion}} \\ \vdash e_{\text{pexpansion}} : \{ \tau_i \}_{i < n} \rightarrow \tau_{\text{expand}} \\ \{ \Phi; \Gamma \vdash \hat{e}_i \rightsquigarrow e_i : \tau_i \}_{i < n} \end{array}}{\Phi; \Gamma \vdash \$a(d_{\text{model}}; \{ \hat{e}_i : \tau_i \}_{i < n})^u \rightsquigarrow e_{\text{pexpansion}} \{ e_i \}_{i < n} : \tau_{\text{expand}}}$$

- (1) Lookup
- (2) Model Validation
- (3 + 4) Expansion + Decoding
- (5) Expansion Validation
- (6) Splice Expansion
- (7) Conclusion

The Typed Livelit Calculus

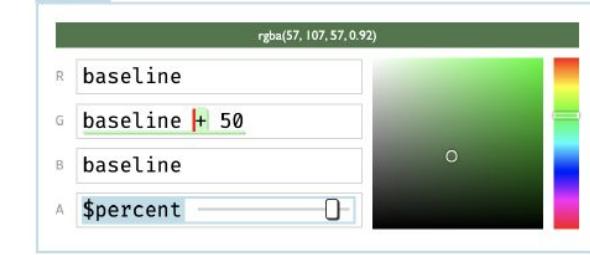
Live Closure Collection (formalized in paper)

1. Replace parameterized expansions with holes
2. Run the program *a la* Hazelnut Live [POPL'19], generating proto-closures
3. Resume any livelit holes that appear in proto-closures to collect livelit closures

Live & Direct & Programmatic

Summary: *Livelits* (a.k.a. *live literals*) in Hazel (hazel.org)

```
let baseline = $slider 0 255 — 0 in  
let $percent = $slider 0 100 in  
let default_color =  
  $color
```



```
let q1_max = 36. in  
let grades =  
  $dataframe
```

A screenshot of a data frame editor interface. At the top, there's a formula bar with the expression "= |q1_max +. 24. +. \$fslider 0. 40. — 0". Below it is a data grid with columns labeled "A1", "A2", "A3", "Midterm", and "Final". The first row has headers: "Andrew", "Cyrus", "David". The second row has values: 80., 61., 75. The third row has values: 92., 64., 81. The fourth row has values: 83.5, 98., 73. The fifth row has values: 95., 70., 82. The sixth row has values: 88., 85., 79. An orange box highlights the value "80." in the "A1" column for Andrew.

	"A1"	"A2"	"A3"	"Midterm"	"Final"
"Andrew"	80.	92.	83.5	95.	88.
"Cyrus"	61.	64.	98.	70.	85.
"David"	75.	81.	73.	82.	79.

Extensible
Persistent
Compositional

Parameterizable
Typed
Live

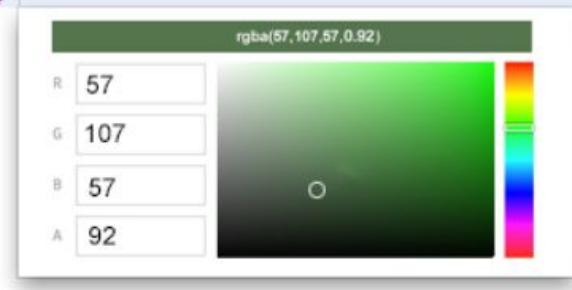
Live & Direct

Prior Work

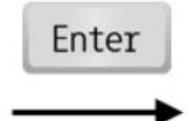
Programmatic

Graphite [Omar et al., ICSE'12]

```
public Color getDefaultColor() {  
    return
```



```
public Color getDefaultColor() {  
    return new Color(  
        57,  
        107,  
        57, 92);
```



Extensible
Persistent
Compositional

Parameterizable
Typed
Live

Live & Direct

Prior Work

Programmatic

mage [Kery et al., UIST'20]

1 mage : user edits table

%common table df			
age	workclass	fnlwgt	
0 90	?	77053	
1 82	Private	132870	
2 66	?	186061	
3 54	Private	140358	
4 41	Private	264663	

2 mage : edits reflect in code

```
# -- generated code --
column_names = list(df)
column_names.pop(6)
column_names.insert(1, "o")
df = df.reindex(columns=column_names)
%common table df
```

age	occupation	workclass
0 90	?	
1 82	Exec-managerial	Priva

Extensible
Persistent
Compositional

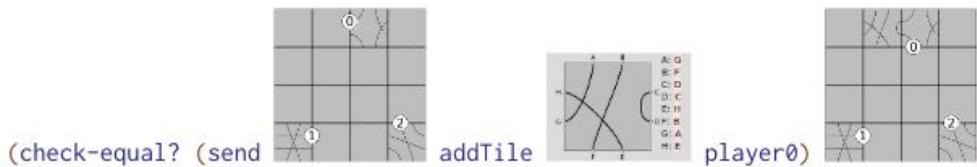
Parameterizable
Typed
Live

Live & Direct

Prior Work →

Programmatic

Interactive syntax in Dr. Racket [Andersen et al., OOPSLA'20]



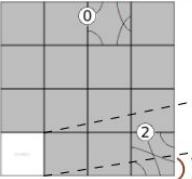
Live & Direct

Prior Work →

Programmatic

Interactive syntax in Dr. Racket [Andersen et al., OOPSLA'20]

```
; Tile -> [Listof Board]
(define (all-possible-configurations t)
  (for/list ([d DEGREES])
    (send t rotate d)))
```



```
(send t rotate d)
```

Extensible
Persistent
Compositional

Parameterizable
Typed
Live

Live & Direct

Prior Work

Programmatic



- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

- + Generic symbolic representations
- + Symbol manipulation affordances
- + Abstraction and composition and calculation and automation

Live & Direct

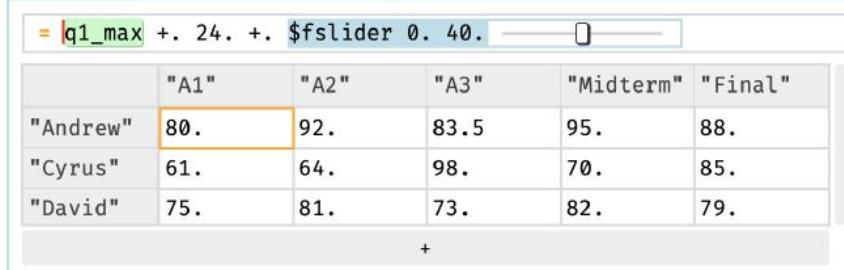
Summary: *Livelits* (a.k.a. *live literals*) in Hazel (hazel.org)

```
let baseline = $slider 0 255 in
let $percent = $slider 0 100 in
let default_color =
  $color
    
```

R baseline
G baseline + 50
B baseline
A \$percent

&

Programmatic

```
let q1_max = 36. in
let grades =
  $dataframe
    = |q1_max +. 24. +. $fslider 0. 40.|
      
```

	"A1"	"A2"	"A3"	"Midterm"	"Final"
"Andrew"	80.	92.	83.5	95.	88.
"Cyrus"	61.	64.	98.	70.	85.
"David"	75.	81.	73.	82.	79.

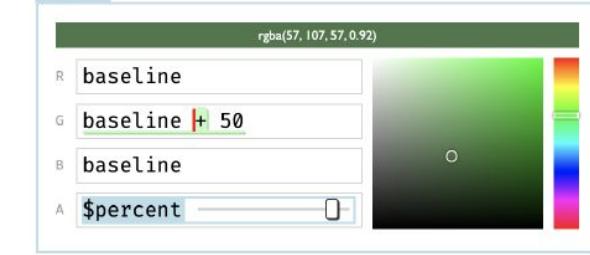
- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

- + Generic symbolic representations
- + Symbol manipulation affordances
- + Abstraction and composition and calculation and automation

Live & Direct & Programmatic

Summary: *Livelits* (a.k.a. *live literals*) in Hazel (hazel.org)

```
let baseline = $slider 0 255 — 0 in
let $percent = $slider 0 100 in
let default_color =
  $color
```



```
let q1_max = 36. in
let grades =
  $dataframe
```

A screenshot of a data frame editor interface. At the top, there's a formula bar with the expression "= |q1_max +. 24. +. \$fslider 0. 40. — 0". Below it is a data table with columns "A1", "A2", "A3", "Midterm", and "Final". The first row has student names: "Andrew", "Cyrus", and "David". The "A1" column contains numerical values: 80., 61., and 75. respectively. The "Midterm" column contains 95. and 70. respectively. The "Final" column contains 88. and 85. respectively. The "A1" cell for Andrew is highlighted with a yellow border.

	"A1"	"A2"	"A3"	"Midterm"	"Final"
"Andrew"	80.	92.	83.5	95.	88.
"Cyrus"	61.	64.	98.	70.	85.
"David"	75.	81.	73.	82.	79.

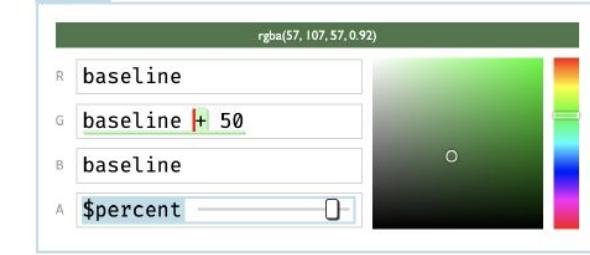
Extensible
Persistent
Compositional

Parameterizable
Typed
Live

Live & Direct & Programmatic

Summary: *Livelits* (a.k.a. *live literals*) in Hazel (hazel.org)

```
let baseline = $slider 0 255 — 0 in  
let $percent = $slider 0 100 in  
let default_color =  
  $color
```



```
let q1_max = 36. in  
let grades =  
  $dataframe
```

	"A1"	"A2"	"A3"	"Midterm"	"Final"
"Andrew"	80.	92.	83.5	95.	88.
"Cyrus"	61.	64.	98.	70.	85.
"David"	75.	81.	73.	82.	79.

Extensible
Persistent
Compositional

Parameterizable
Typed
Live

Thank you!

Live & Direct & *Programmatic*



- + Specialized representations
- + Direct manipulation affordances
- + Live feedback
- + Generic symbolic representations
- + Symbol manipulation affordances
- + Abstraction and composition and calculation and automation