Filling Typed Holes with Live GUIs

STRUMENTA

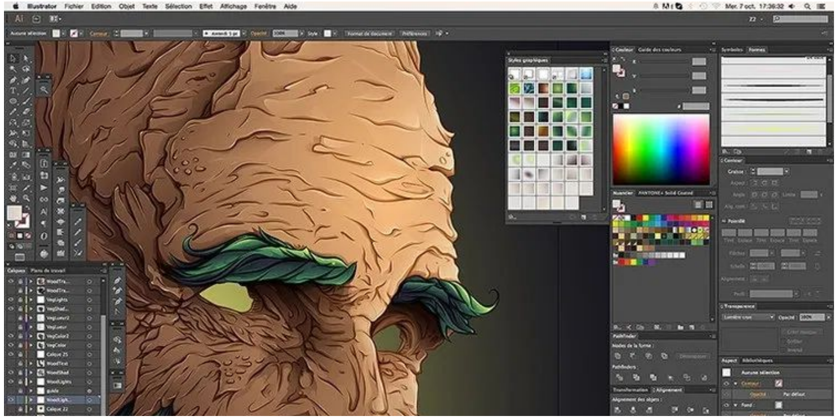
Cyrus Omar, David Moon, Andrew Blinn
Future of Programming Lab (FP Lab) @ University of Michigan

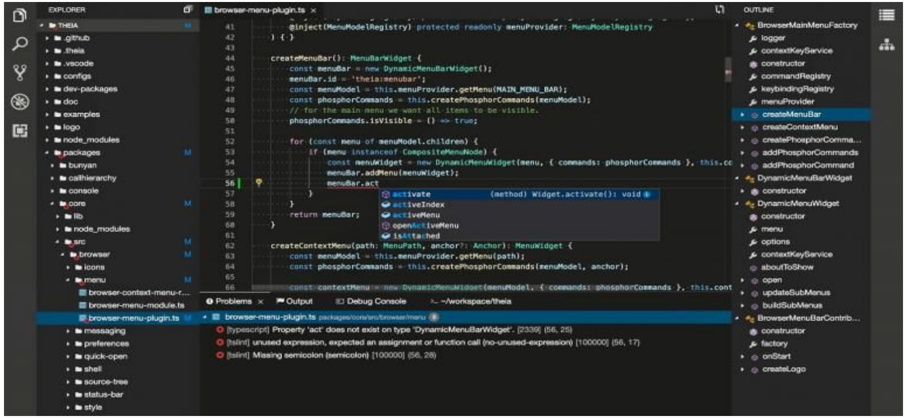
Ian Voysey
Carnegie Mellon University

Nick Collins, Ravi Chugh
University of Chicago

Creative End Users

Programmers


Illustrator Fichier Edition Objet Texte Sélection Effet Affichage Fenêtre Aide
Mer. 7 oct. 17:36:32



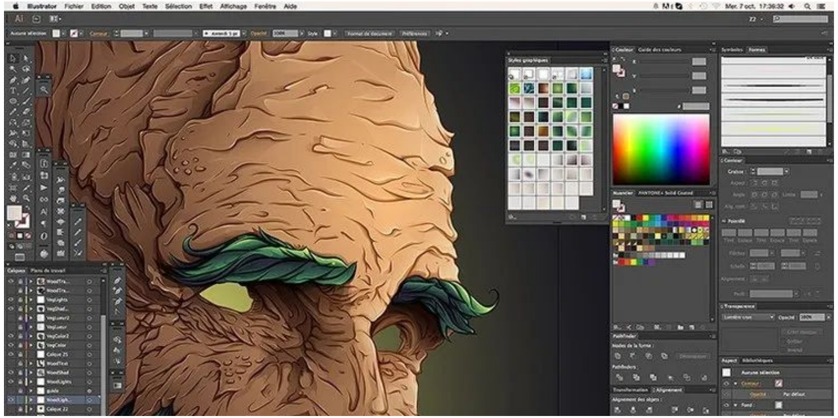
```
        @inject(MenuModelRegistry) protected readonly menuProvider: MenuModelRegistry
    ) { }

    createMenuBar(): MenuBarWidget {
        const menuBar = new DynamicMenuBarWidget();
        menuBar.id = 'theia:menubar';
        const menuModel = this.menuProvider.getMenu(MAIN_MENU_BAR);
        const phosphorCommands = this.createPhosphorCommands(menuModel);
        // for the main menu we want all items to be visible.
        phosphorCommands.isVisible = () => true;

        for (const menu of menuModel.children) {
            if (menu instanceof CompositeMenuNode) {
                const menuWidget = new DynamicMenuWidget(menu, { commands: phosphorCommands }, this.co
                menuBar.addMenu(menuWidget);
                menuBar.act
            }
        }
        return menuBar;
    }

    createContextMenu(path: MenuPath, anchor?: Anchor): MenuWidget {
        const menuModel = this.menuProvider.getMenu(path);
        const phosphorCommands = this.createPhosphorCommands(menuModel, anchor);

        const contextMenu = new DynamicMenuWidget(menuModel, { commands: phosphorCommands }, this.cont
```

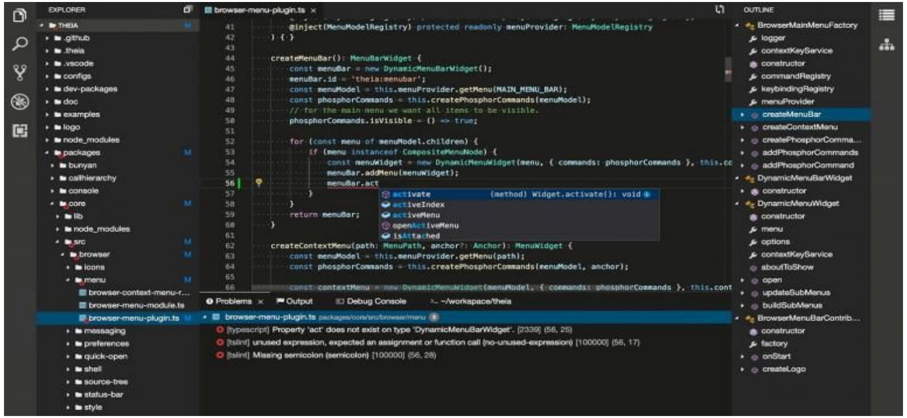
browser-menu-plugin.ts packages/core/src/browser/menu 3

```
[typescript] Property 'act' does not exist on type 'DynamicMenuBarWidget'. [2339] (56, 25)
[tslint] unused expression, expected an assignment or function call (no-unused-expression) [100000] (56, 17)
[tslint] Missing semicolon (semicolon) [100000] (56, 28)
```

Live & Direct

Abstract & Symbolic





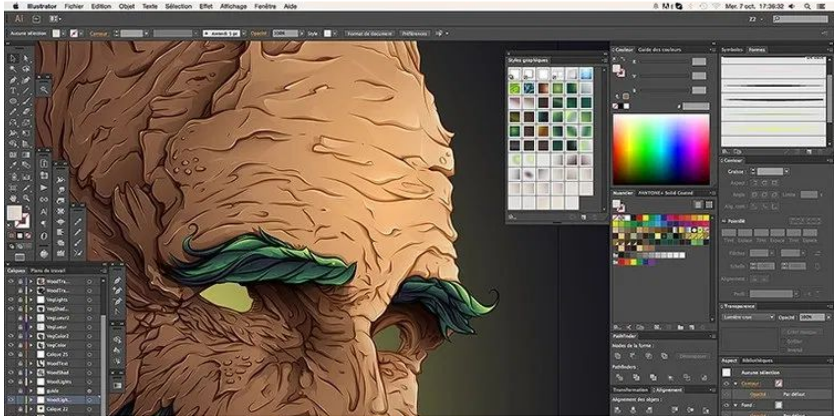
```
    @inject(MenuModelRegistry) protected readonly menuProvider: MenuModelRegistry
) { }

createMenuBar(): MenuBarWidget {
    const menuBar = new DynamicMenuBarWidget();
    menuBar.id = 'theia:menubar';
    const menuModel = this.menuProvider.getMenu(MAIN_MENU_BAR);
    const phosphorCommands = this.createPhosphorCommands(menuModel);
    // for the main menu we want all items to be visible.
    phosphorCommands.isVisible = () => true;

    for (const menu of menuModel.children) {
        if (menu instanceof CompositeMenuNode) {
            const menuWidget = new DynamicMenuWidget(menu, { commands: phosphorCommands }, this.co
            menuBar.addMenu(menuWidget);
            menuBar.act
        }
    }
    return menuBar;
}

createContextMenu(path: MenuPath, anchor?: Anchor): MenuWidget {
    const menuModel = this.menuProvider.getMenu(path);
    const phosphorCommands = this.createPhosphorCommands(menuModel, anchor);

    const contextMenu = new DynamicMenuWidget(menuModel, { commands: phosphorCommands }, this.cont
```

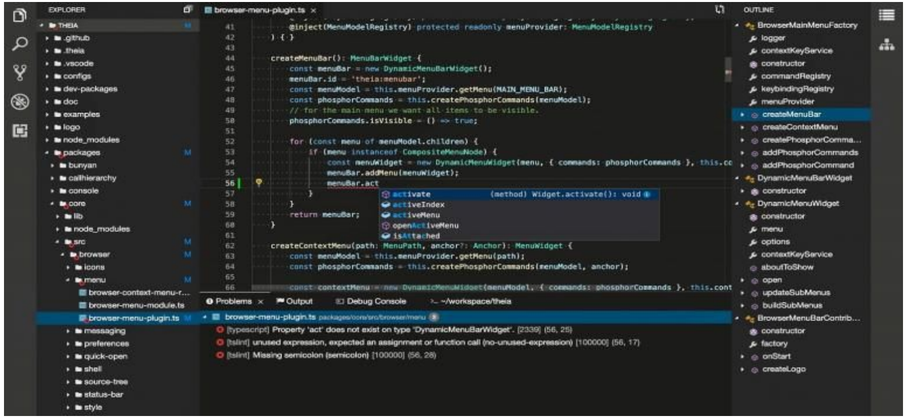
```
activate          (method) Widget.activate(): void
activeIndex
activeMenu
openActiveMenu
isAttached
```

```
browser-menu-plugin.ts packages/core/src/browser/menu 3
[typescript] Property 'act' does not exist on type 'DynamicMenuBarWidget'. [2339] (56, 25)
[tslint] unused expression, expected an assignment or function call (no-unused-expression) [100000] (56, 17)
[tslint] Missing semicolon (semicolon) [100000] (56, 28)
```

Live & Direct

Abstract & Symbolic





```
        @inject(MenuModelRegistry) protected readonly menuProvider: MenuModelRegistry
    ) { }

    createMenuBar(): MenuBarWidget {
        const menuBar = new DynamicMenuBarWidget();
        menuBar.id = 'theia:menubar';
        const menuModel = this.menuProvider.getMenu(MAIN_MENU_BAR);
        const phosphorCommands = this.createPhosphorCommands(menuModel);
        // for the main menu we want all items to be visible.
        phosphorCommands.isVisible = () => true;

        for (const menu of menuModel.children) {
            if (menu instanceof CompositeMenuNode) {
                const menuWidget = new DynamicMenuWidget(menu, { commands: phosphorCommands }, this.co
                menuBar.addMenu(menuWidget);
                menuBar.act
            }       activate                    (method) Widget.activate(): void
        }           activeIndex
        return menuBar;  activeMenu
    }               openActiveMenu
                    isAttached
    createContextMenu(path: MenuPath, anchor?: Anchor): MenuWidget {
        const menuModel = this.menuProvider.getMenu(path);
        const phosphorCommands = this.createPhosphorCommands(menuModel, anchor);

        const contextMenu = new DynamicMenuWidget(menuModel, { commands: phosphorCommands }, this.cont
```

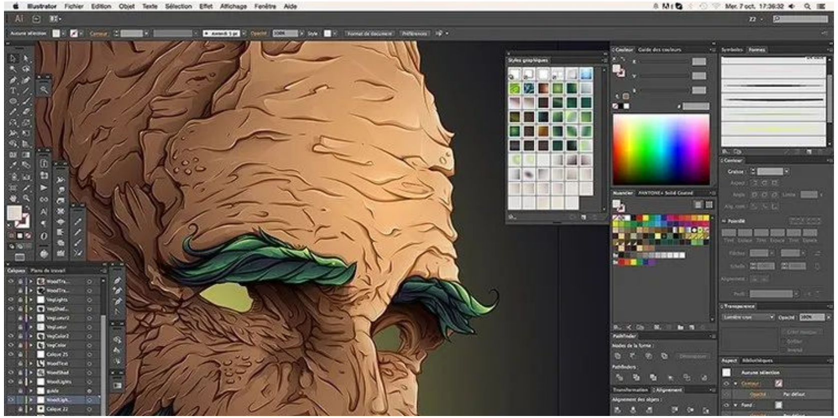
Problems × | Output | Debug Console | ~/workspace/theia

browser-menu-plugin.ts packages/core/src/browser/menu 3

```
[typescript] Property 'act' does not exist on type 'DynamicMenuBarWidget'. [2339] (56, 25)
[tslint] unused expression, expected an assignment or function call (no-unused-expression) [100000] (56, 17)
[tslint] Missing semicolon (semicolon) [100000] (56, 28)
```

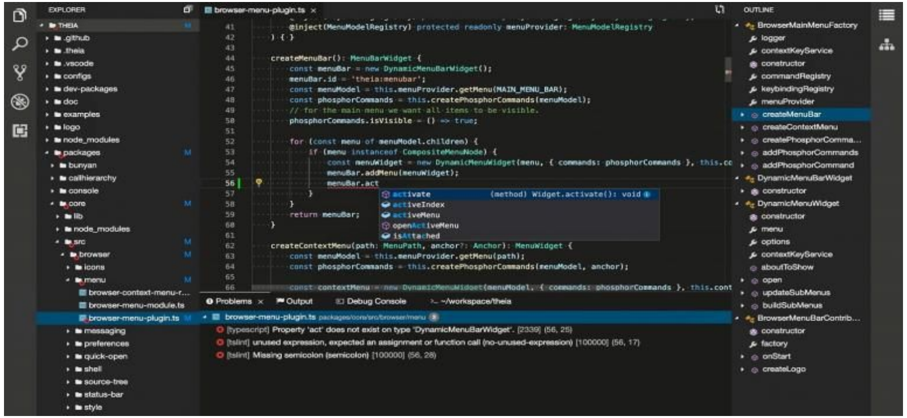
- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

Live & Direct



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

Abstract & Symbolic



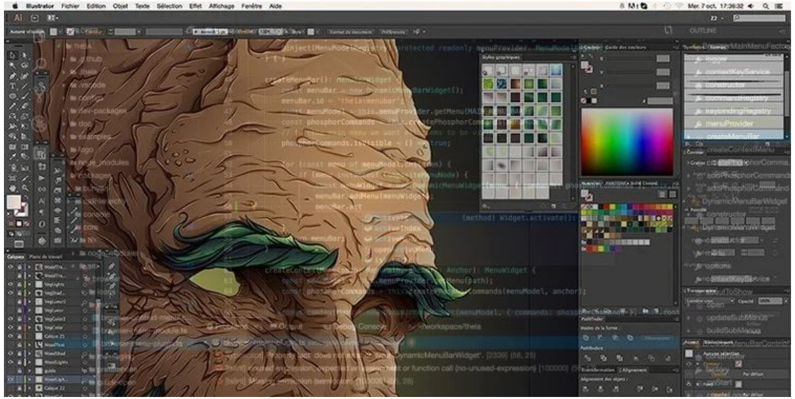
```
        @inject(MenuModelRegistry) protected readonly menuProvider: MenuModelRegistry
    ) { }

    createMenuBar(): MenuBarWidget {
        const menuBar = new DynamicMenuBarWidget();
        menuBar.id = 'theia:menubar';
        const menuModel = this.menuProvider.getMenu(MAIN_MENU_BAR);
        const phosphorCommands = this.createPhosphorCommands(menuModel);
        // for the main menu we want all items to be visible.
        phosphorCommands.isVisible = () => true;

        for (const menu of menuModel.children) {
            if (menu instanceof CompositeMenuNode) {
                const menuWidget = new DynamicMenuWidget(menu, { commands: phosphorCommands }, this.co
                menuBar.addMenu(menuWidget);
                menuBar.act
            }                activate                    (method) Widget.activate(): void
        }                    activeIndex
        return menuBar;      activeMenu
    }                        openActiveMenu
                             isAttached
    createContextMenu(path: MenuPath, anchor?: Anchor): MenuWidget {
        const menuModel = this.menuProvider.getMenu(path);
        const phosphorCommands = this.createPhosphorCommands(menuModel, anchor);

        const contextMenu = new DynamicMenuWidget(menuModel, { commands: phosphorCommands }, this.cont
```

Problems × Output Debug Console ~/workspace/theia

```
browser-menu-plugin.ts packages/core/src/browser/menu 3
[typescript] Property 'act' does not exist on type 'DynamicMenuBarWidget'. [2339] (56, 25)
[tslint] unused expression, expected an assignment or function call (no-unused-expression) [100000] (56, 17)
[tslint] Missing semicolon (semicolon) [100000] (56, 28)
```

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

Live & Direct & Abstract & Symbolic

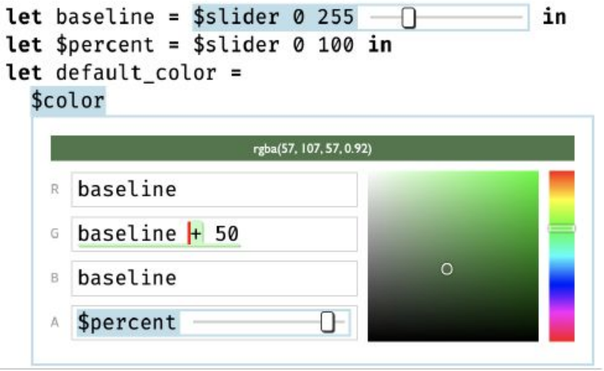


- Specialized representations
- Direct manipulation affordances
- Live (immediate + uninterrupted) feedback

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

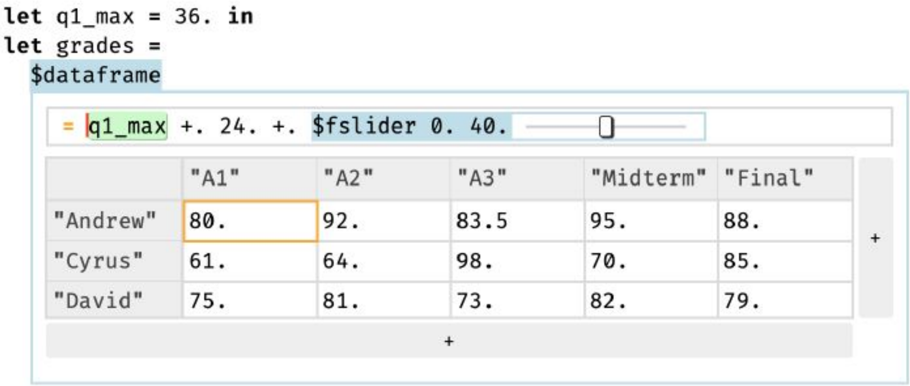
Live & Direct & Abstract & Symbolic

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



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

```
baseline
baseline + 50
baseline
$percent
```



```
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 & Abstract & Symbolic

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

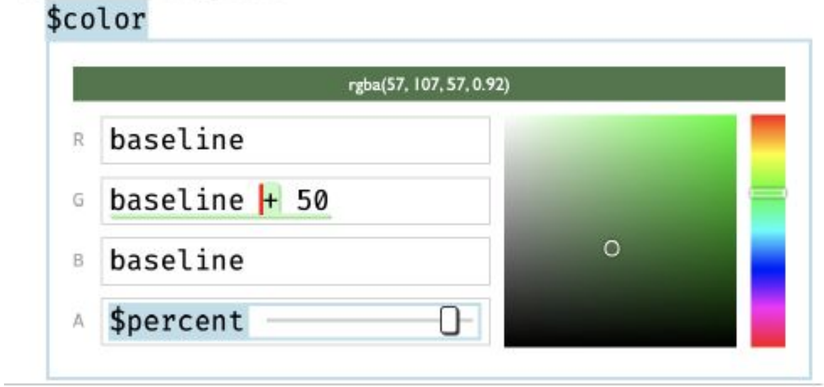
Live & Direct Demo! [hazel.org/livelits]

- 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



$color

rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

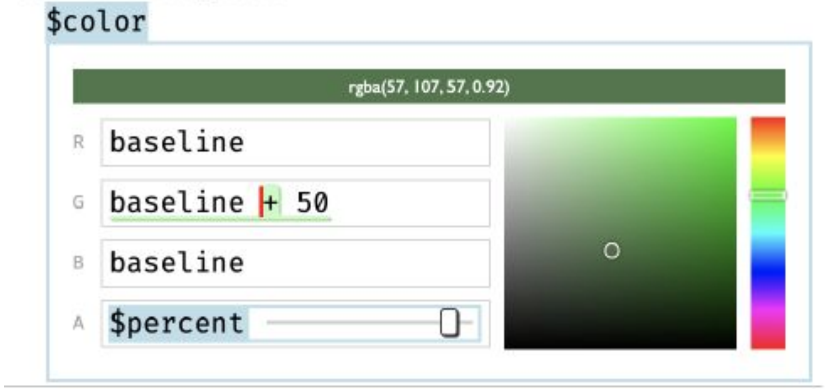
B baseline

A $percent

```
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



$color

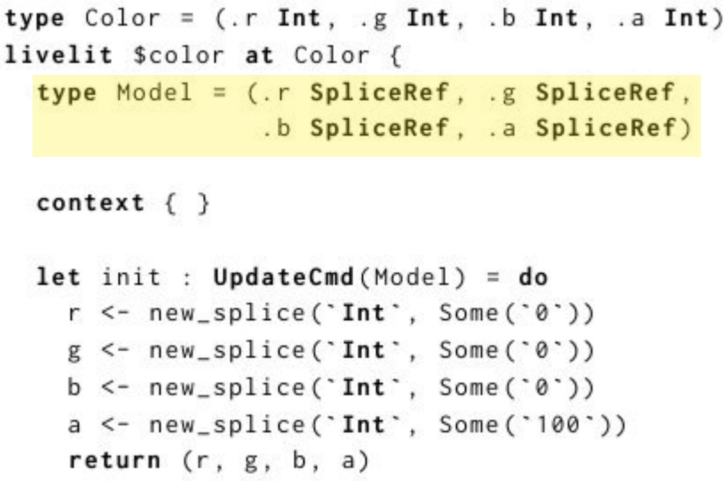
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

A $percent

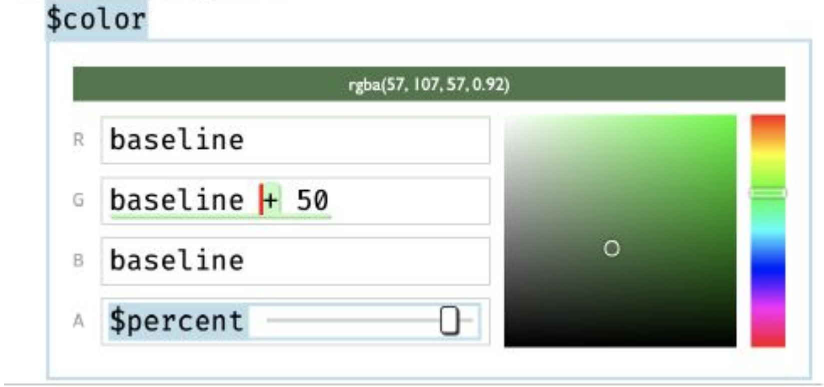


```
type Color = (.r Int, .g Int, .b Int, .a Int)
livelit $color at Color {
  type Model = (.r SpliceRef, .g SpliceRef,
                .b SpliceRef, .a SpliceRef)

  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)
```

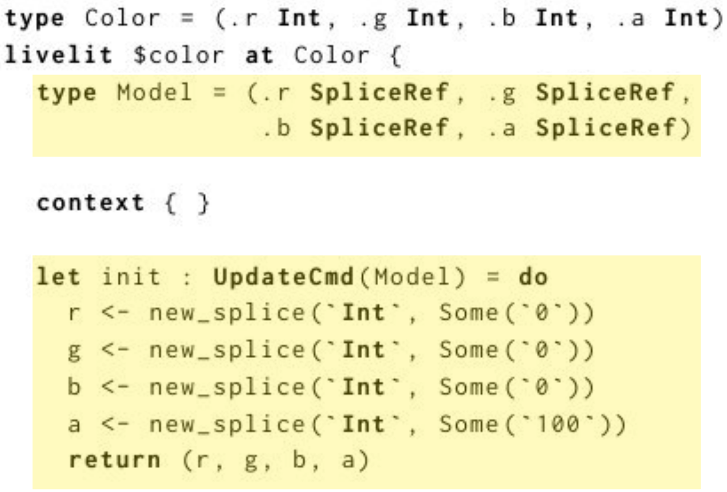
...

Live & Direct & Abstract & Symbolic

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



```
baseline
baseline + 50
baseline
$percent
```

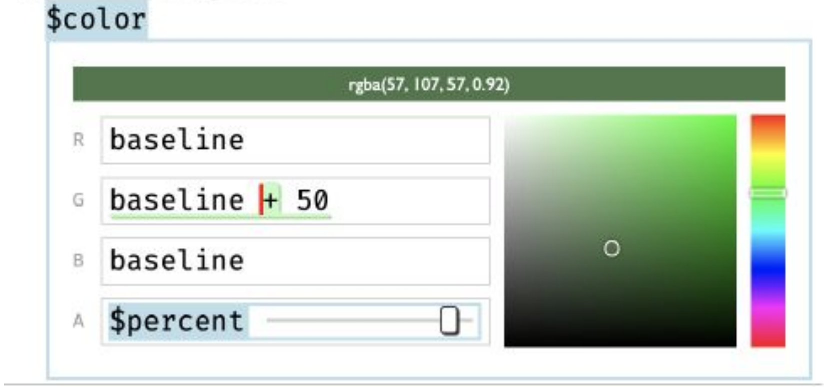


```
type Color = (.r Int, .g Int, .b Int, .a Int)
livelit $color at Color {
  type Model = (.r SpliceRef, .g SpliceRef,
                .b SpliceRef, .a SpliceRef)

  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



$color

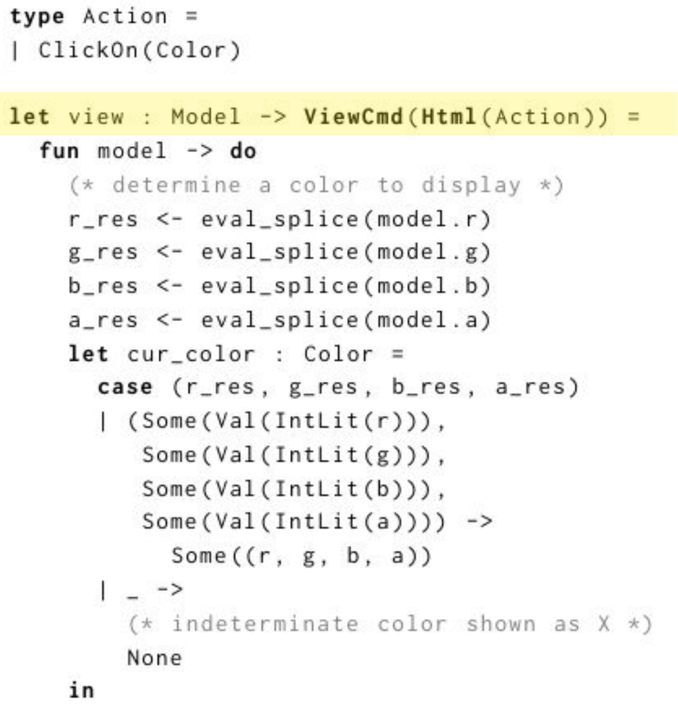
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

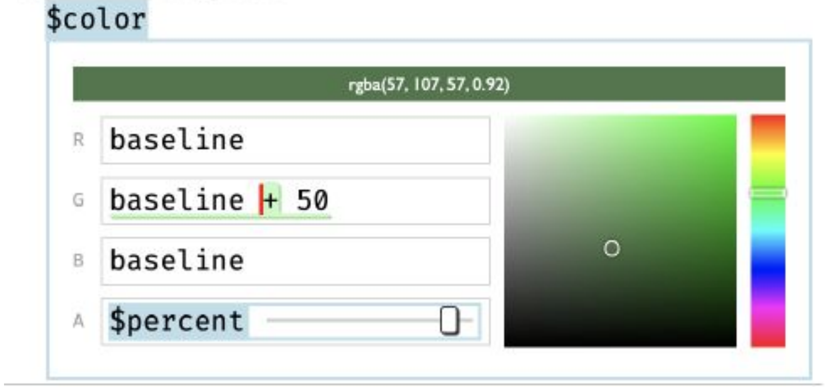
A $percent



```
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


$color

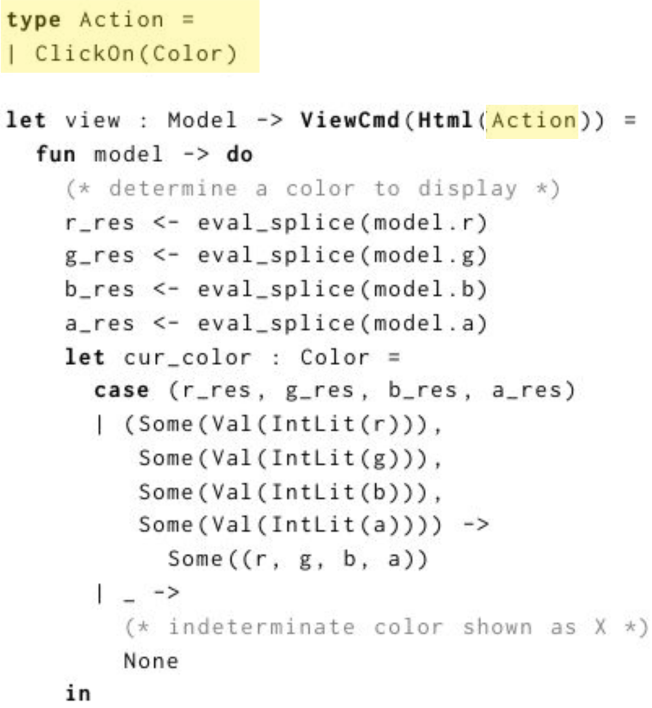
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

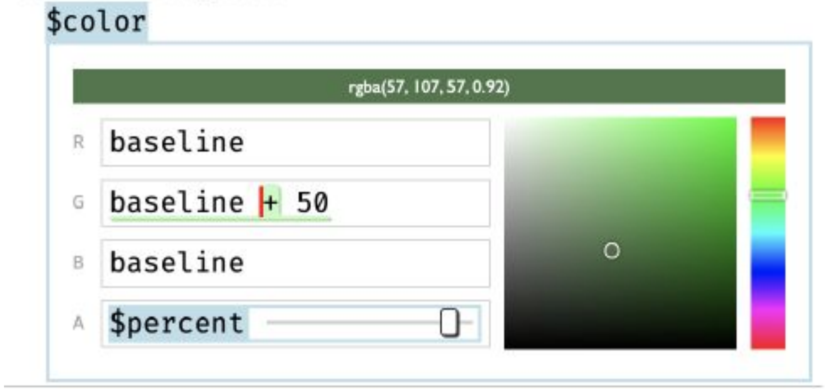
B baseline

A $percent


```
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



$color

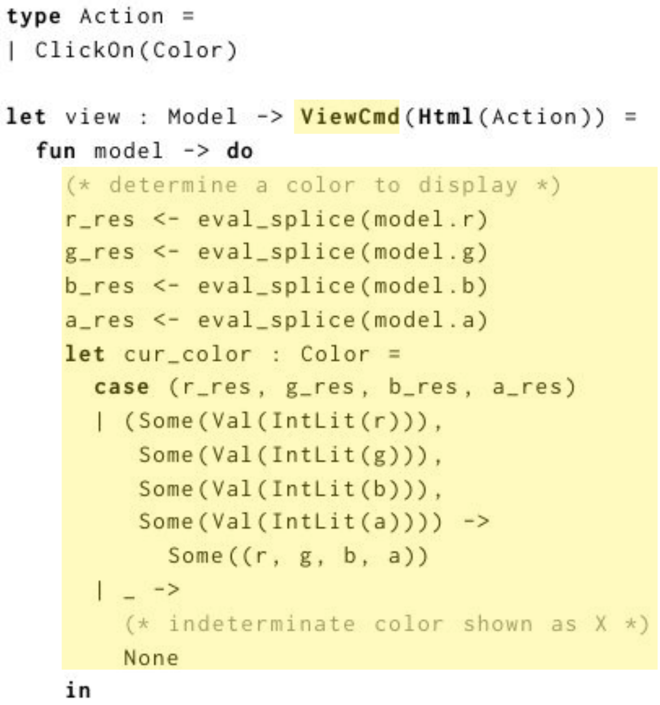
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

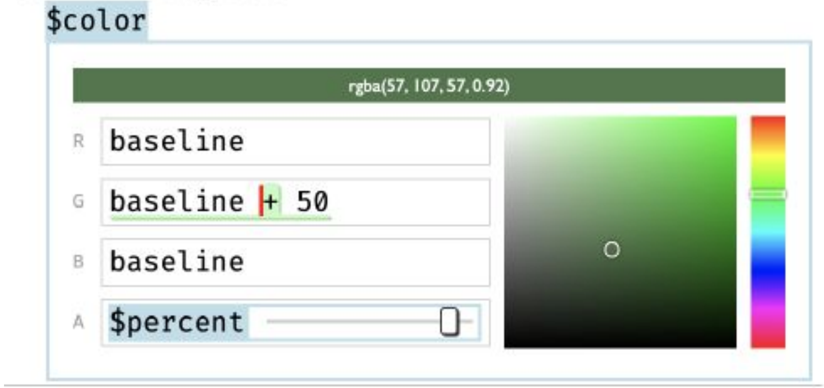
A $percent



```
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



$color

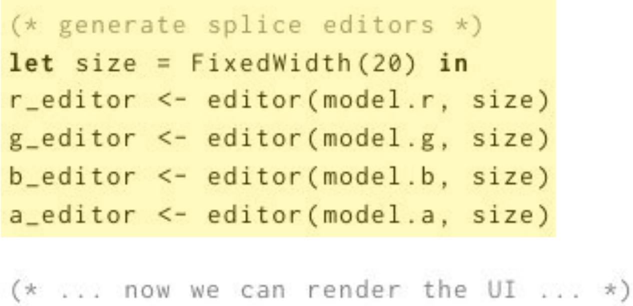
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

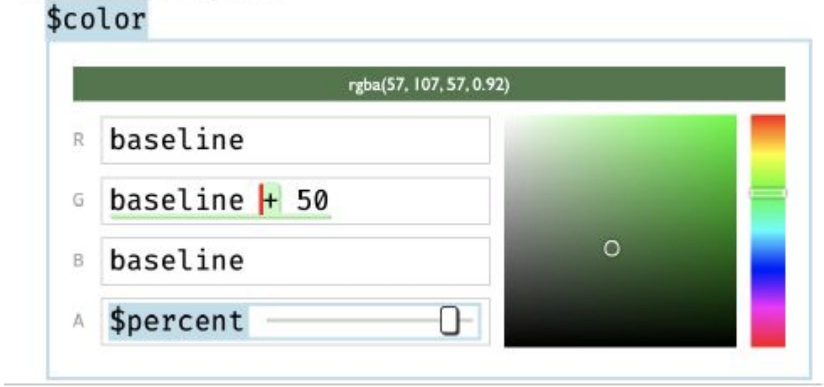
A $percent



```
(* 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



$color

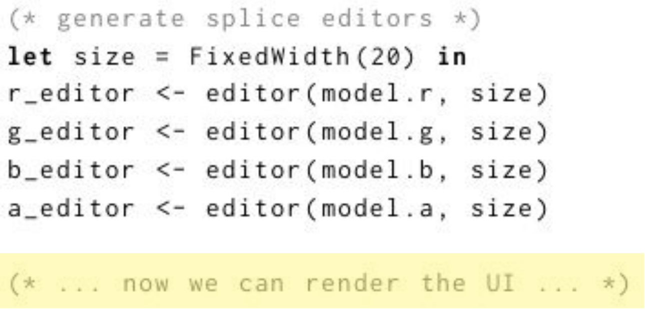
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

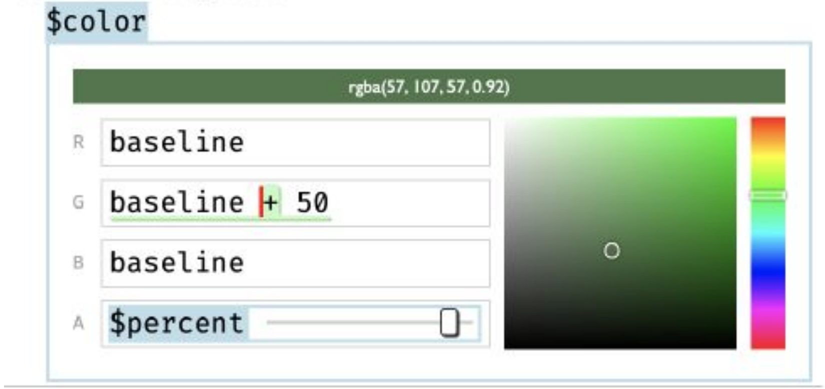
A $percent



```
(* 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



$color

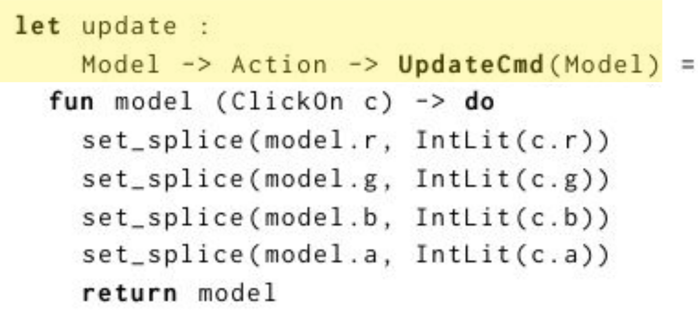
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

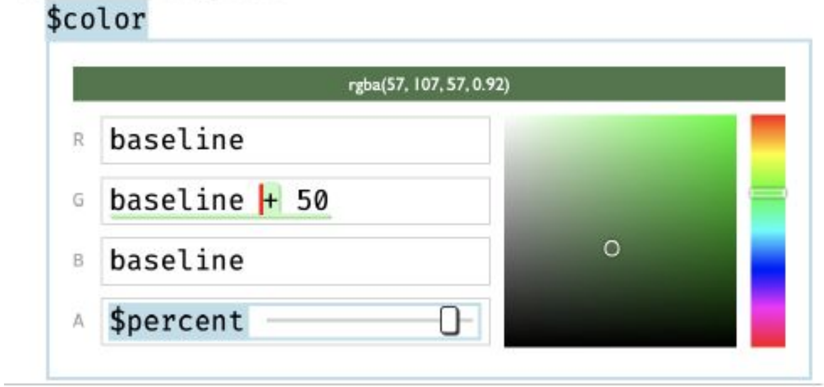
A $percent



```
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



$color

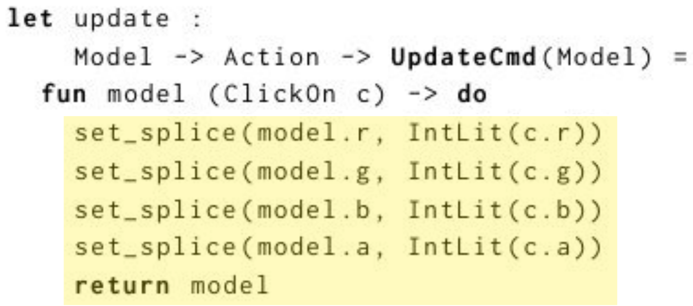
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

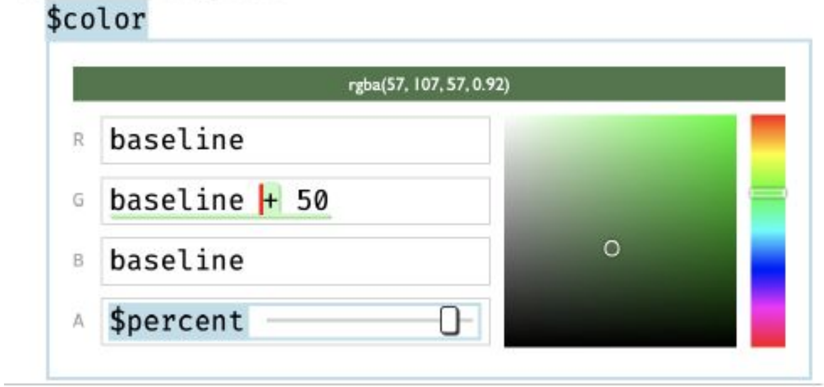
A $percent



```
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



$color

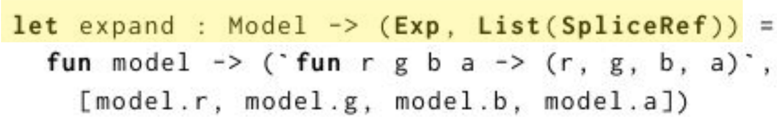
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

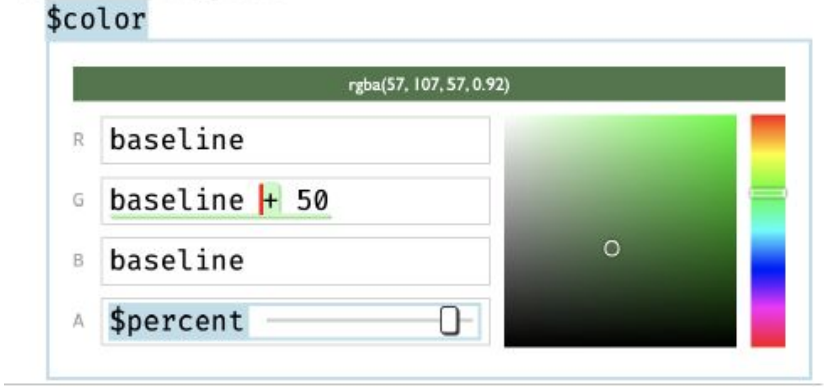
A $percent



```
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



$color

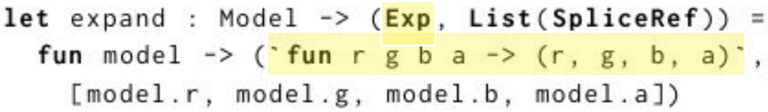
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

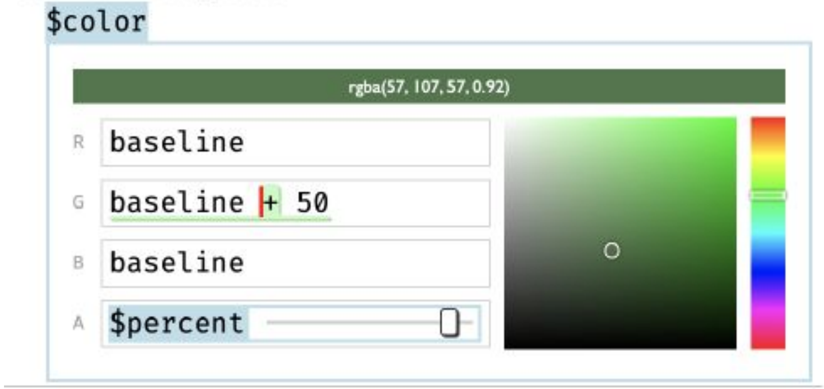
A $percent



```
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

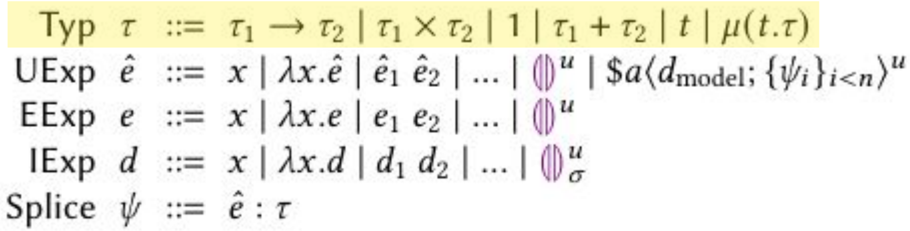


```
$color
rgba(57, 107, 57, 0.92)
R baseline
G baseline + 50
B baseline
A $percent
```

```
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

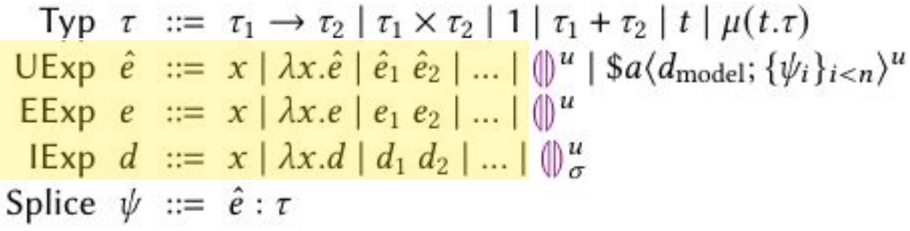
Overview



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

The Typed Livelit Calculus

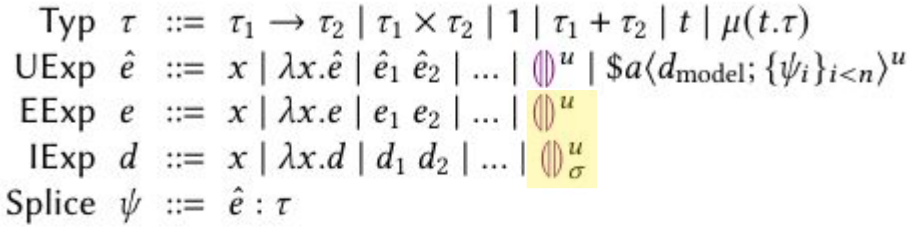
Overview



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

The Typed Livelit Calculus

Overview

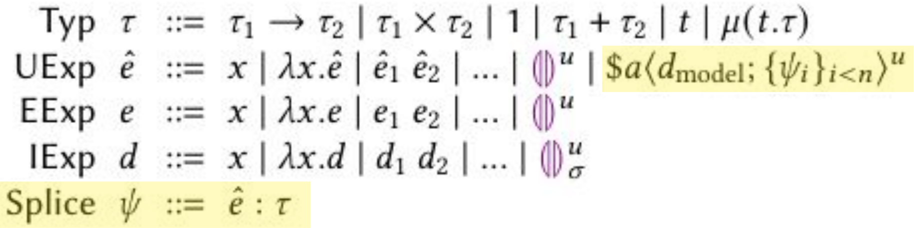


	Typ	τ	$::=$	$\tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \tau_1 + \tau_2 \mid t \mid \mu(t.\tau)$
Unexpanded	UExp	$\hat{e}$	$::=$	$x \mid \lambda x.\hat{e} \mid \hat{e}_1\ \hat{e}_2 \mid \ldots \mid \llparenthesis\rrparenthesis^u \mid \$a\langle d_{\text{model}}; \{\psi_i\}_{i<n}\rangle^u$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

Elaboration
[Hazelnut Live, POPL’19]

The Typed Livelit Calculus

Overview

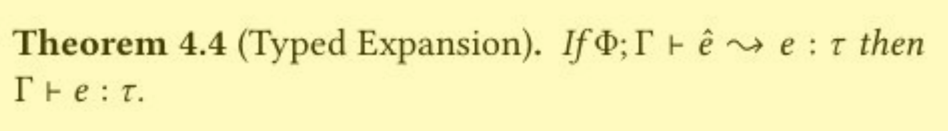


	Typ	τ	$::=$	$\tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \tau_1 + \tau_2 \mid t \mid \mu(t.\tau)$
Unexpanded	UExp	$\hat{e}$	$::=$	$x \mid \lambda x.\hat{e} \mid \hat{e}_1\ \hat{e}_2 \mid \ldots \mid \llparenthesis\rrparenthesis^u \mid \$a\langle d_{\text{model}}; \{\psi_i\}_{i<n}\rangle^u$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

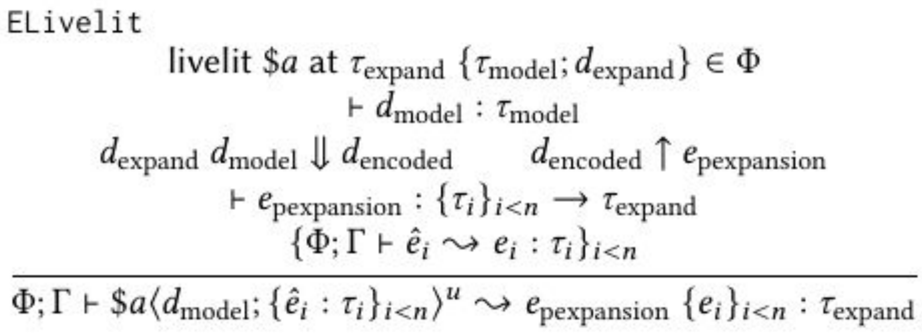
Expansion

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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

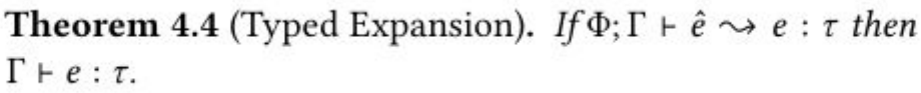


`ELivelit`

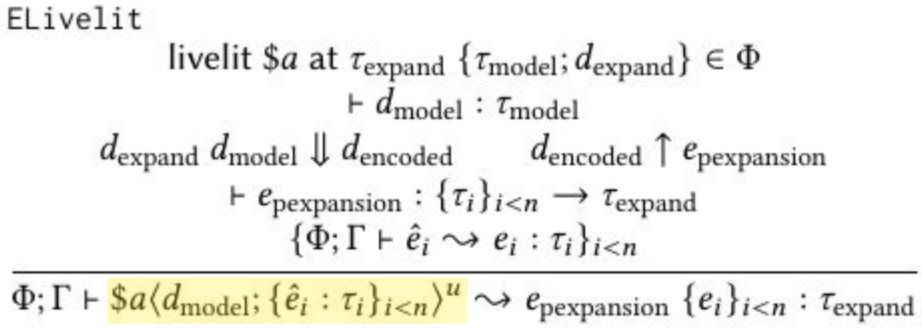
$$\frac{\begin{array}{c} \text{livelit } \$a \text{ at } \tau_{\text{expand}}\ \{\tau_{\text{model}}; d_{\text{expand}}\} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}}\ d_{\text{model}} \Downarrow d_{\text{encoded}} \qquad d_{\text{encoded}} \uparrow e_{\text{pexpansion}}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

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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

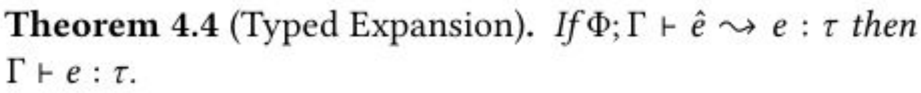


`ELivelit`

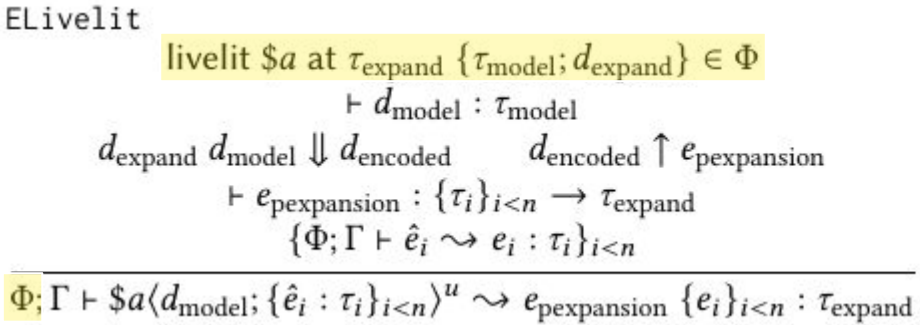
$$\frac{\begin{array}{c}\text{livelit } \$a \text{ at } \tau_{\text{expand}}\ \{\tau_{\text{model}}; d_{\text{expand}}\} \in \Phi \\ \vdash d_{\text{model}} : \tau_{\text{model}} \\ d_{\text{expand}}\ d_{\text{model}} \Downarrow d_{\text{encoded}} \qquad d_{\text{encoded}} \Uparrow e_{\text{pexpansion}}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

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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



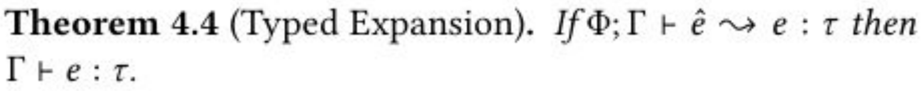
ELivelit

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

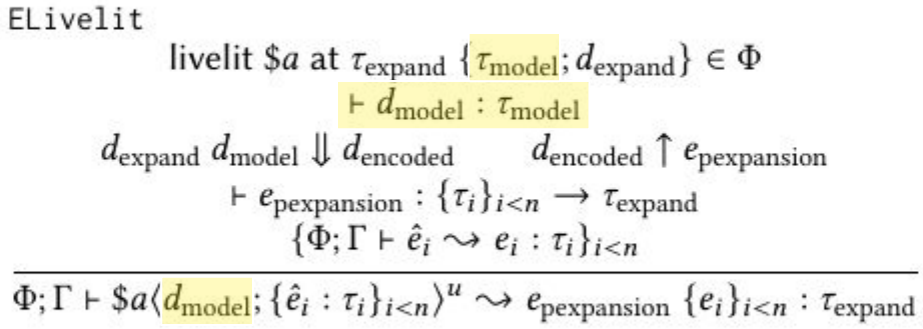
(1) Lookup

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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



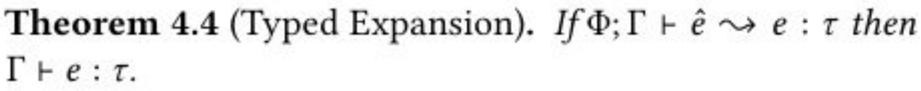
ELivelit

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

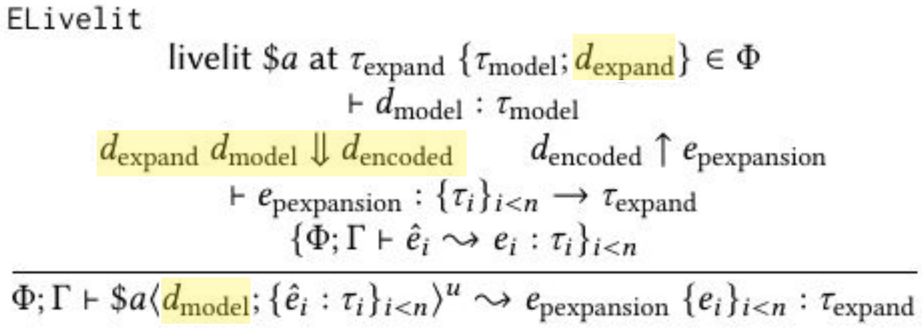
(2) Model Validation

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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



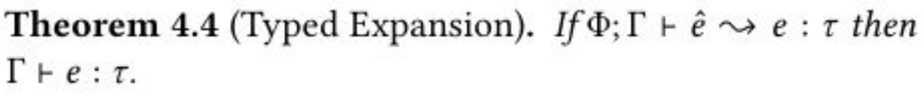
ELivelit

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

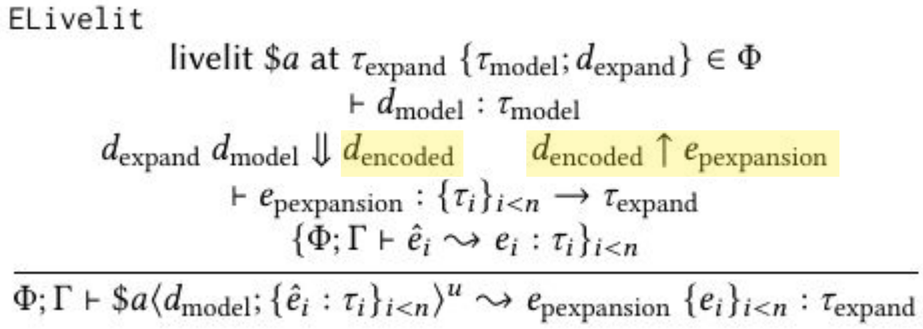
(3) Expansion Generation

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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



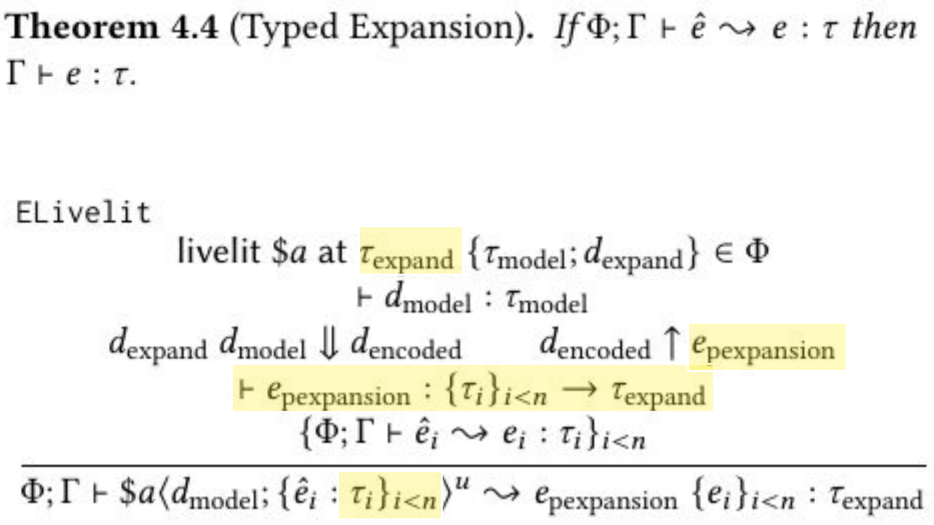
ELivelit

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

(4) Decoding

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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

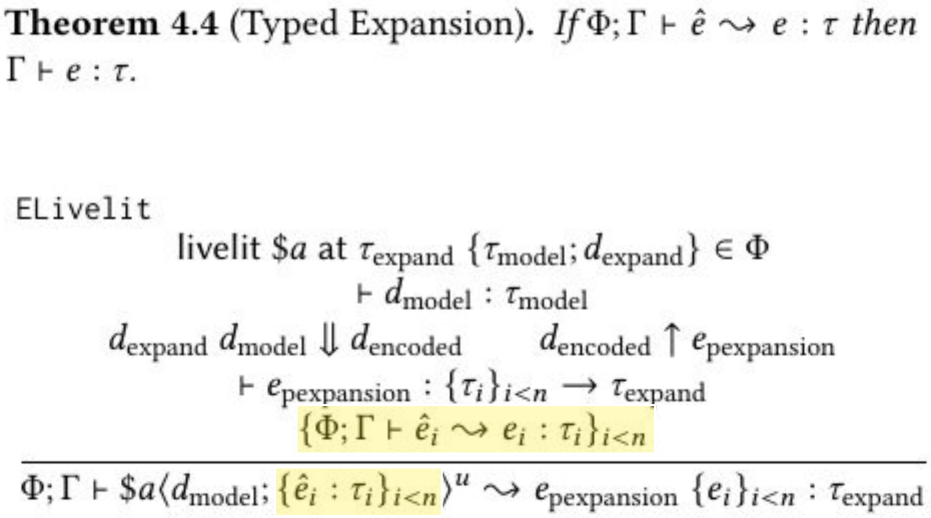
`ELivelit`

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

(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* $\Gamma \vdash e : \tau$.

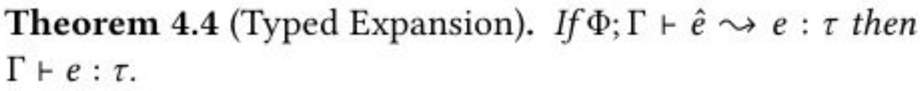
ELivelit

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

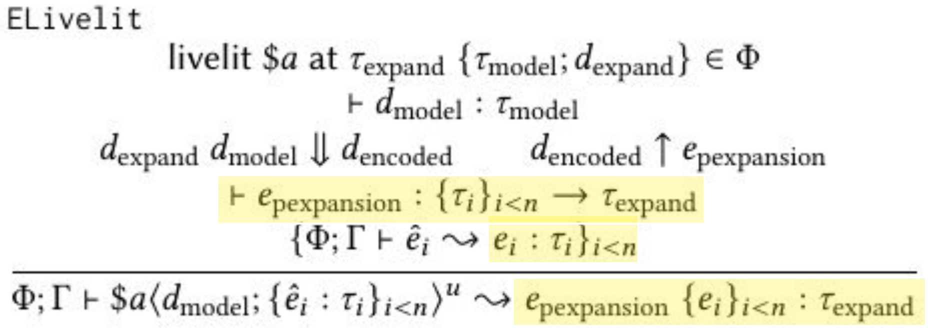
(6) Splice Expansion

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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



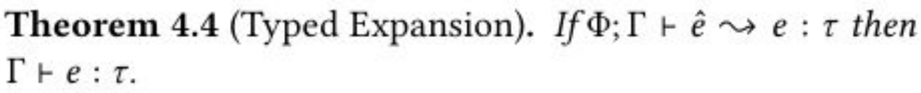
ELivelit

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

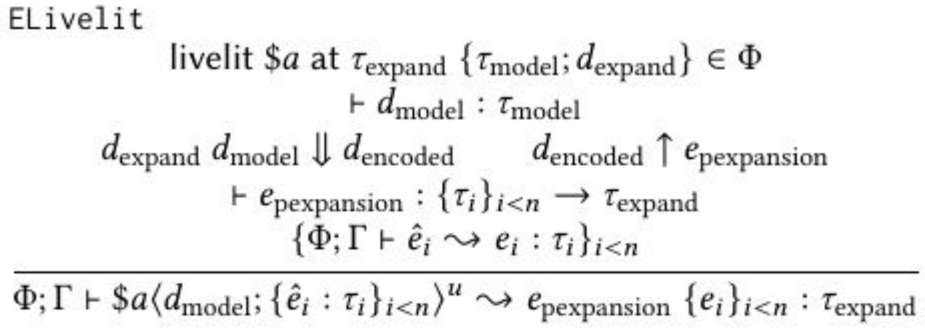
(7) Conclusion

The Typed Livelit Calculus

Expansion (Mechanized in Agda)



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



ELivelit

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

(1) Lookup

(2) Model Validation

(3+ 4) Expansion + Decoding

(5) Expansion Validation

(6) Splice Expansion

(7) Conclusion

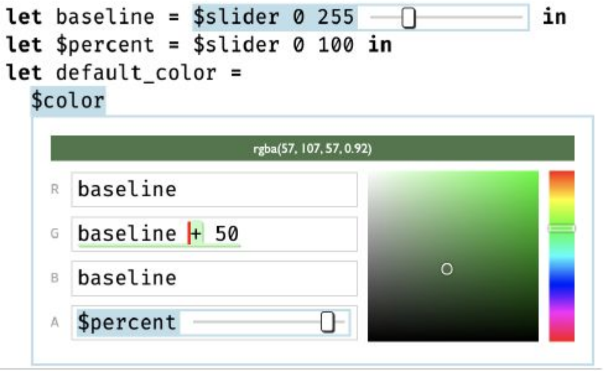
The Typed Livelit Calculus

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

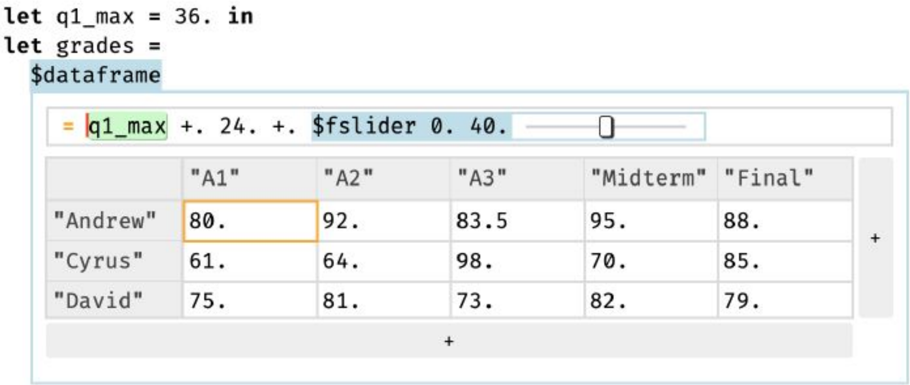
rgba(57, 107, 57, 0.92)

R baseline

G baseline + 50

B baseline

A $percent



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

= q1_max +. 24. +. $fslider 0. 40.

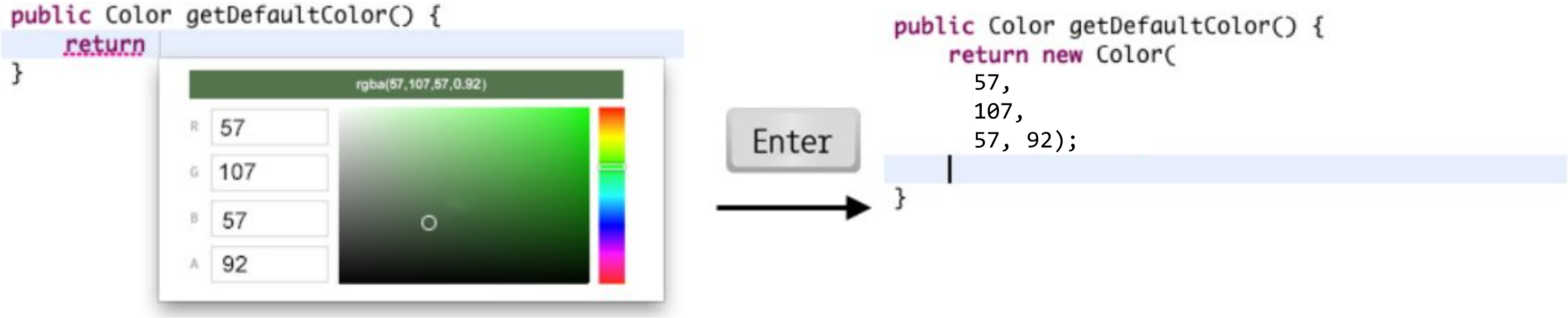
	"A1"	"A2"	"A3"	"Midterm"	"Final"
"Andrew"	80.	92.	83.5	95.	88.
"Cyrus"	61.	64.	98.	70.	85.
"David"	75.	81.	73.	82.	79.

Extensible
Persistent
Compositional

Parameterizable
Typed
Live

Live & Direct → Prior Work → Programmatic

Graphite [Omar et al., ICSE'12]



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

R 57
G 107
B 57
A 92

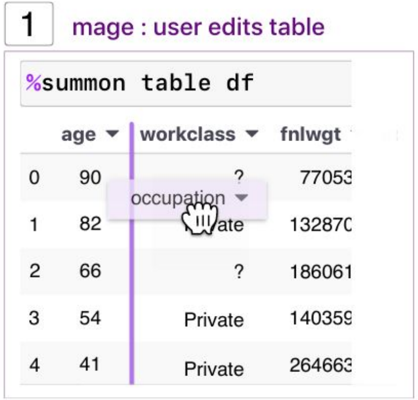
```
public Color getDefaultColor() {
    return new Color(
      57,
      107,
      57, 92);

}
```

Extensible
~~Persistent~~
~~Compositional~~

~~Parameterizable~~
Typed
~~Live~~

Live & Direct —Prior Work→ Programmatic

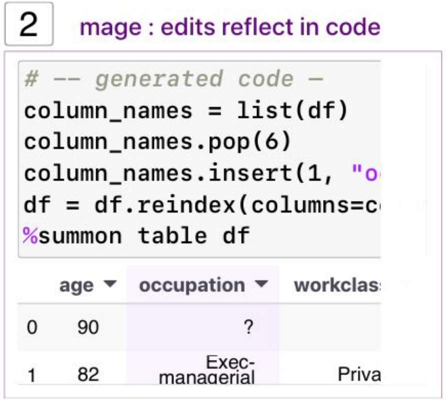
mage [Kery et al., UIST'20]



1 mage : user edits table

```
%summon table df
```

	age	workclass	fnlwgt
0	90	?	77053
1	82	Private	132870
2	66	?	186061
3	54	Private	140359
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=c
%summon table df
```

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

Extensible

Persistent

~~Compositional~~

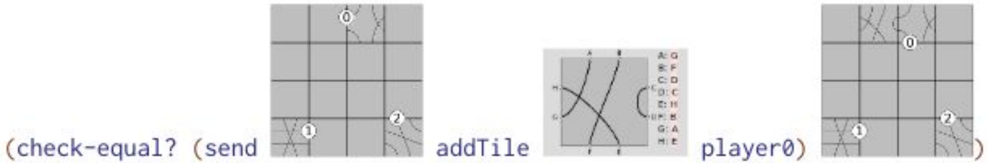
~~Parameterizable~~

~~Typed~~

~~Live~~

Live & Direct → Prior Work → Programmatic

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

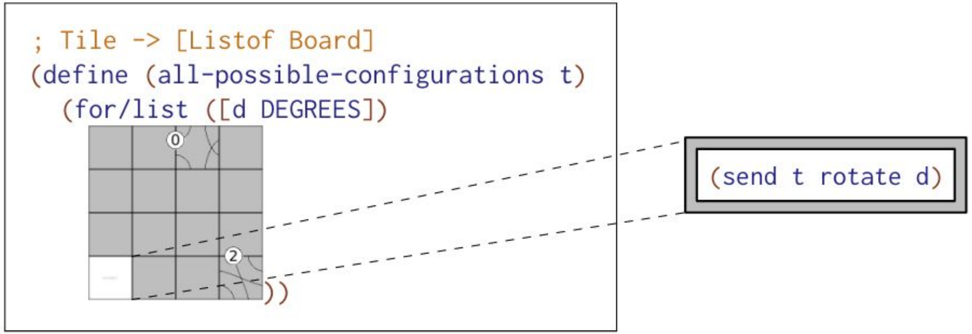


```
(check-equal? (send [board] addTile [tile] player0) [board])
```

Live & Direct → Programmatic

Prior Work

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)
```

Extensible

Persistent

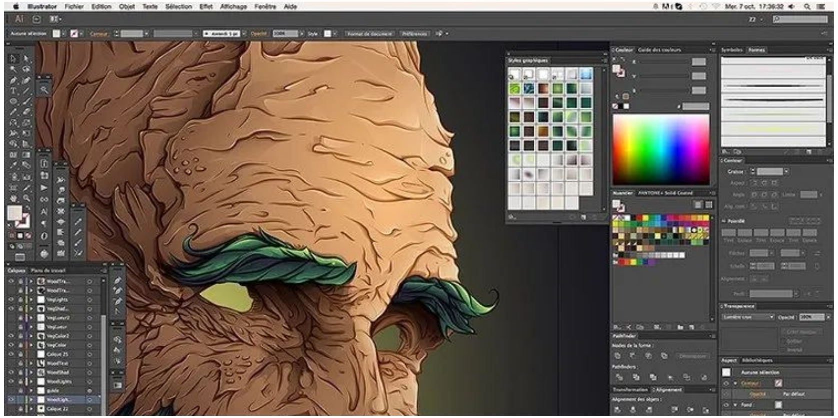
~~Compositional~~

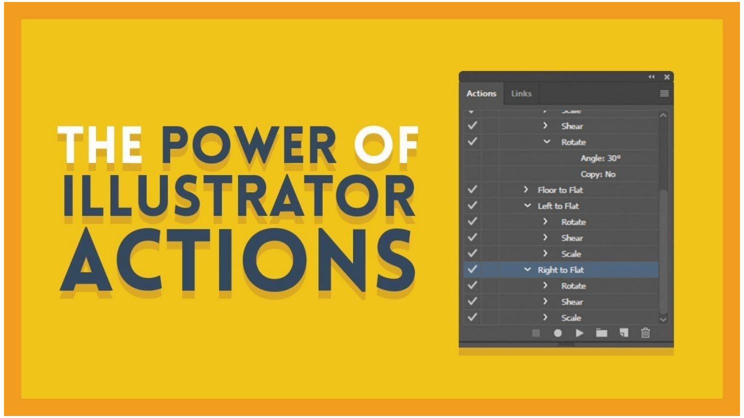
~~Parameterizable~~

~~Typed~~

~~Live~~

Live & Direct ← Prior Work — Programmatic



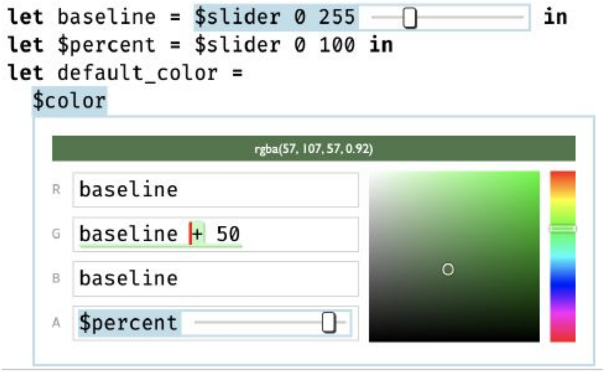


- + Specialized representations
- + Direct manipulation affordances
- + Live (immediate + uninterrupted) feedback

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

Live & Direct & *Programmatic*

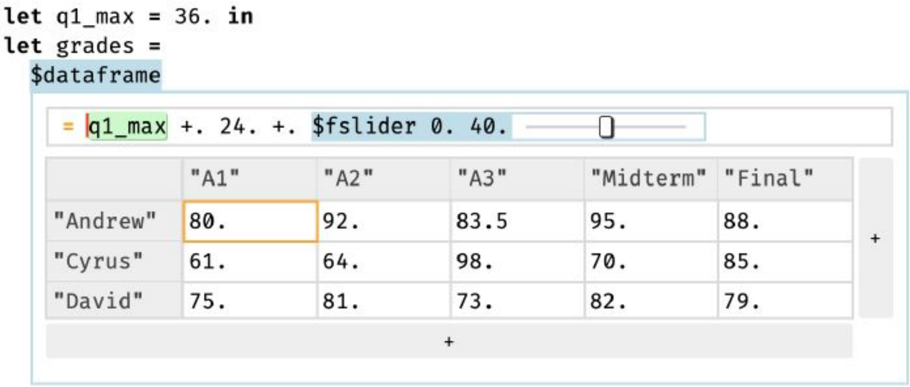
Summary: *Livelits* (a.k.a. *live literals*) in Hazel (hazel.org)



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

rgba(57, 107, 57, 0.92)

R	baseline
G	baseline + 50
B	baseline
A	$percent



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

```
= q1_max +. 24. +. $fslider 0. 40.
```

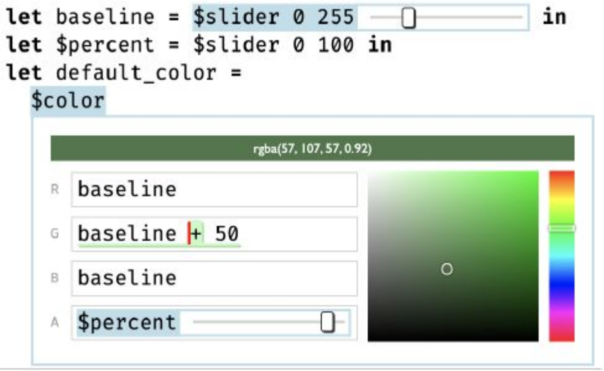
	"A1"	"A2"	"A3"	"Midterm"	"Final"
"Andrew"	80.	92.	83.5	95.	88.
"Cyrus"	61.	64.	98.	70.	85.
"David"	75.	81.	73.	82.	79.

- + 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

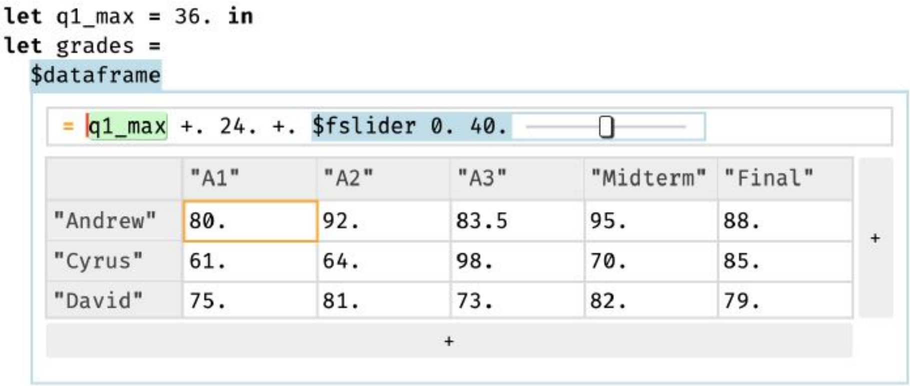
Summary: *Livelits* (a.k.a. *live literals*) in Hazel (hazel.org)



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

rgba(57, 107, 57, 0.92)

R	baseline
G	baseline + 50
B	baseline
A	$percent



```
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.

Extensible

Persistent

Compositional

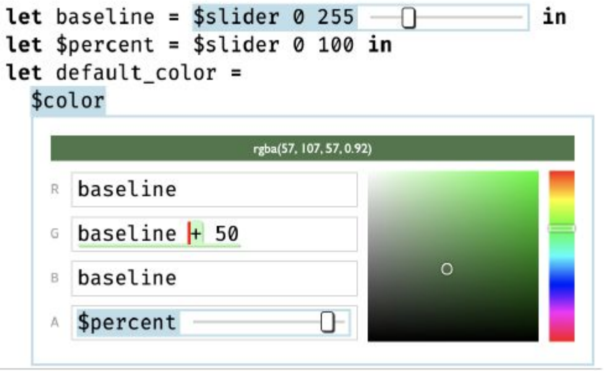
Parameterizable

Typed

Live

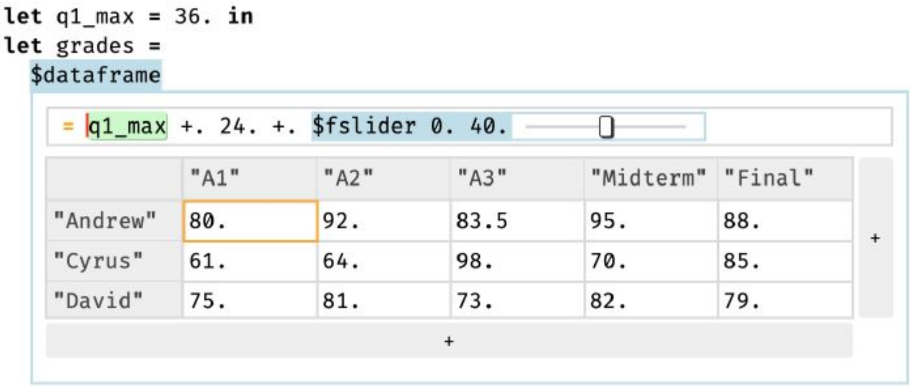
Live & Direct & *Programmatic*

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



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

R	baseline
G	baseline + 50
B	baseline
A	$percent



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

```
= q1_max +. 24. +. $fslider 0. 40.
```

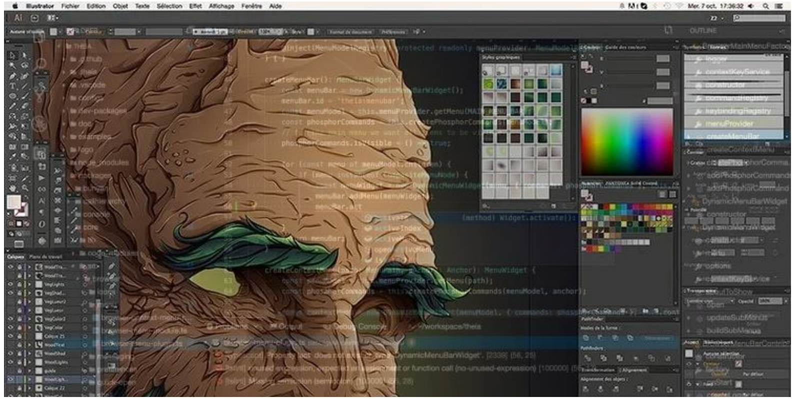
	"A1"	"A2"	"A3"	"Midterm"	"Final"
"Andrew"	80.	92.	83.5	95.	88.
"Cyrus"	61.	64.	98.	70.	85.
"David"	75.	81.	73.	82.	79.

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