Evaluation: the TAL-0 abstract machine

- the abstract machine contains the code and data.
- an evaluation step changes the state (code and data) of the abstract machine.

- A register file R maps each register r to some value (integer or label) $R(r)$.

$$R ::= \{\mathsf{r1} \mapsto \nu_1, \ldots, \mathsf{rk} \mapsto \nu_k\}$$

(each ν_i is a value)

- For TAL-0, the only heap values are instruction sequences.

$$h ::= I$$

Extensions of TAL-0 will need to consider other kinds of heap values.

- A heap H is a partial map: H maps some labels l to heap values $H(l)$.

$$H ::= \{l_1 \mapsto h_1, \ldots l_m \mapsto h_m\}$$

An abstract machine state consists of a heap, a register file and the current sequence being executed.

$$M ::= (H, R, I)$$

The previous example has three instruction sequences

$I_1 = \mathsf{r3} := \mathsf{0}; \mathsf{r2} := \mathsf{r1};\ \mathsf{jump\ loop}$

$I_2 = \mathsf{if\ r1\ jump\ done}; \mathsf{r3} := \mathsf{r2} + \mathsf{r3}; \mathsf{r1} := \mathsf{r1} + -\mathsf{1};\ \mathsf{jump\ loop}$

$I_3 =\ \mathsf{jump\ r4}$

We have the heap $H_0 = \{\mathsf{prod} \mapsto I_1, \mathsf{loop} \mapsto I_2, \mathsf{done} \mapsto I_3\}$.

The starting state of the machine is supposed to be of the form

$$M_0 = (H_0, R_0, I_1)$$

where $R_0(\mathsf{r1}) = \mathsf{n}$ is an integer and $R_0(\mathsf{r4})$ is a label.

A possible execution sequence: ...

$H_0,$	$\{r1 \mapsto \mathsf{2},$	$r2 \mapsto \mathsf{0},$	$r3 \mapsto \mathsf{0},$	$r4 \mapsto \mathsf{l}\},$	I_1
$H_0,$	$\{r1 \mapsto \mathsf{2},$	$r2 \mapsto \mathsf{0},$	$r3 \mapsto \mathsf{0},$	$r4 \mapsto \mathsf{l}\},$	$\mathsf{r2} := \mathsf{r1};\ \mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{2},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{0}$	$r4 \mapsto \mathsf{l}\},$	$\mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{2},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{0}$	$r4 \mapsto \mathsf{l}\},$	I_2
$H_0,$	$\{r1 \mapsto \mathsf{2},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{0}$	$r4 \mapsto \mathsf{l}\},$	$r3 := r2 + r3; \mathsf{r1} := \mathsf{r1} + -\mathsf{1};\ \mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{2},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{2}$	$r4 \mapsto \mathsf{l}\},$	$\mathsf{r1} := \mathsf{r1} + -\mathsf{1};\ \mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{1},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{2}$	$r4 \mapsto \mathsf{l}\},$	$\mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{1},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{2}$	$r4 \mapsto \mathsf{l}\},$	I_2
$H_0,$	$\{r1 \mapsto \mathsf{1},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{2}$	$r4 \mapsto \mathsf{l}\},$	$r3 := r2 + r3; \mathsf{r1} := \mathsf{r1} + -\mathsf{1};\ \mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{1},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{4}$	$r4 \mapsto \mathsf{l}\},$	$\mathsf{r1} := \mathsf{r1} + -\mathsf{1};\ \mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{0},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{4}$	$r4 \mapsto \mathsf{l}\},$	$\mathsf{jump\ loop}$
$H_0,$	$\{r1 \mapsto \mathsf{0},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{4}$	$r4 \mapsto \mathsf{l}\},$	I_2
$H_0,$	$\{r1 \mapsto \mathsf{0},$	$r2 \mapsto \mathsf{2},$	$r3 \mapsto \mathsf{4}$	$r4 \mapsto \mathsf{l}\},$	$\mathsf{jump}\ r4$

As usual, we formalize this using evaluation rules.

As usual, we formalize this using evaluation rules.

$$\frac{H(\hat{R}(\nu)) = I}{(H, R, \ \mathsf{jump}\ \nu) \longrightarrow (H, R, I)} \text{ (E-Jump)}$$

where the lookup function $\hat{R}$ returns the value corresponding to an operand:

$$\hat{R}(r) = R(r)$$
$$\hat{R}(n) = n$$
$$\hat{R}(l) = l$$

The JUMP instruction loads a new instruction sequence which should then be executed.

The machine is stuck if $\hat{R}(\nu)$ is not a label, or if the label does not correspond to some instruction sequence in the heap.

Otherwise, we consume one instruction from the current instruction sequence.

The MOV and ADD instructions modify the register file.

$$(H, R, r_d := \nu; I) \longrightarrow (H, R \oplus \{r_d \mapsto \hat{R}(\nu)\}, I) \qquad \text{(E-Mov)}$$

Otherwise, we consume one instruction from the current instruction sequence.

The MOV and ADD instructions modify the register file.

$$(H, R, r_d := \nu; I) \longrightarrow (H, R \oplus \{r_d \mapsto \hat{R}(\nu)\}, I) \qquad \text{(E-Mov)}$$

$$\frac{R(r_s) = n_1 \qquad \hat{R}(\nu) = n_2}{(H, R, r_d := r_s + \nu; I) \longrightarrow (H, R \oplus \{r_d \mapsto n_1 + n_2\}, I)} \text{(E-Add)}$$

(The machine is stuck in the second case if $R(r_s)$ or $\hat{R}(\nu)$ is not an integer.)

The conditional jump instruction either loads a new instruction sequence or just consumes one instruction.

$$\frac{R(r) = \mathsf{0} \qquad H(\hat{R}(\nu)) = I'}{(H, R, \mathsf{if}\ r\ \mathsf{jump}\ \nu; I) \longrightarrow (H, R, I')}\ \text{(E-IfEq)}$$

The conditional jump instruction either loads a new instruction sequence or just consumes one instruction.

$$\frac{R(r) = \mathsf{0} \qquad H(\hat{R}(\nu)) = I'}{(H, R, \mathsf{if}\ r\ \mathsf{jump}\ \nu; I) \longrightarrow (H, R, I')}\ \text{(E-IfEq)}$$

$$\frac{R(r) = n \qquad n \neq \mathsf{0}}{(H, R, \mathsf{if}\ r\ \mathsf{jump}\ \nu; I) \longrightarrow (H, R, I)}\ \text{(E-IfNeq)}$$

(The machine is stuck if $R(r)$ is not an integer or, in the first case, if $\hat{R}(\nu)$ is not a label.)

Consider the following simple code:

$$\begin{aligned} \mathsf{l}: \quad & \mathsf{r1} := \mathsf{5}; \\ & \mathsf{jump}\ \mathsf{r1} \end{aligned}$$

Consider the following simple code:

```
l :   r1 := 5;
      jump r1
```

Define instruction sequence $I = \mathsf{r1} := \mathsf{5};\ \mathsf{jump}\ \mathsf{r1}$ and heap $H = \{\mathsf{l} \mapsto I\}$.

Corresponding to the above code, starting with register file $R = \{\mathsf{r1} \mapsto \mathsf{0}\}$ we have the evaluation step

$$(H, \{\mathsf{r1} \mapsto \mathsf{0}\}, I) \longrightarrow (H, \{\mathsf{r1} \mapsto \mathsf{5}\},\ \mathsf{jump}\ \mathsf{r1})$$

Consider the following simple code:

```
l :   r1 := 5;
      jump r1
```

Define instruction sequence $I = \mathsf{r1} := \mathsf{5};\ \mathsf{jump\ r1}$ and heap $H = \{\mathsf{l} \mapsto I\}$.

Corresponding to the above code, starting with register file $R = \{\mathsf{r1} \mapsto \mathsf{0}\}$ we have the evaluation step

$$(H, \{\mathsf{r1} \mapsto \mathsf{0}\}, I) \longrightarrow (H, \{\mathsf{r1} \mapsto \mathsf{5}\},\ \mathsf{jump\ r1})$$

The machine is now stuck: no further evaluation step is possible because r1 stores an integer instead of a label.

Consider the following simple code:

```
l :   r1 := 5;
      jump r1
```

Define instruction sequence $I = \mathsf{r1} := \mathsf{5};\ \mathsf{jump}\ \mathsf{r1}$ and heap $H = \{\mathsf{l} \mapsto I\}$.

Corresponding to the above code, starting with register file $R = \{\mathsf{r1} \mapsto \mathsf{0}\}$ we have the evaluation step

$$(H, \{\mathsf{r1} \mapsto \mathsf{0}\}, I) \longrightarrow (H, \{\mathsf{r1} \mapsto \mathsf{5}\},\ \mathsf{jump}\ \mathsf{r1})$$

The machine is now stuck: no further evaluation step is possible because r1 stores an integer instead of a label.

Hence to filter out such bad programs, we need to introduce typing rules.

Initial idea for a TAL-0 typing system: introduce two different types Int and Code for integers and labels.

In the previous example, we will start with the register file type $\Gamma = \{\mathsf{r1} : \mathsf{Int}\}$.

After the instruction $\mathsf{r1} = \mathsf{5}$ the register file type remains the same.

Then the second instruction jump r1 fails to type check because $\Gamma(\mathsf{r1})$ is Int instead of Code.

Hence the code is rejected, as desired.

Initial idea for a TAL-0 typing system: introduce two different types Int and Code for integers and labels.

In the previous example, we will start with the register file type $\Gamma = \{\mathsf{r1} : \mathsf{Int}\}$.

After the instruction $\mathsf{r1} = \mathsf{5}$ the register file type remains the same.

Then the second instruction $\mathsf{jump}\ \mathsf{r1}$ fails to type check because $\Gamma(\mathsf{r1})$ is Int instead of Code.

Hence the code is rejected, as desired.

Is this idea enough?

Consider the following code:

$$\begin{aligned} \mathsf{l} : \quad & \mathsf{r1} := \mathsf{5}; \\ & \mathsf{r2} := \mathsf{l}'; \\ & \mathsf{jump}\ \mathsf{r2} \end{aligned}$$

Label l' points to some other instruction sequence I'.

$I = \mathsf{r1} := \mathsf{5}; \mathsf{r2} := \mathsf{l}';\ \mathsf{jump}\ \mathsf{r2}$ and heap $H = \{\mathsf{l} : I, \mathsf{l}' \mapsto I'\}$.

Should the above code be well-typed? After the first two instructions, the register file type will be $\{\mathsf{r1} : \mathsf{Int}, \mathsf{r2} : \mathsf{Code}\}$, as it should be.

Answer: depends on I'...

Consider the code

$$\mathsf{l}' : \quad \mathsf{jump}\ \mathsf{r1};$$

Clearly the instruction sequence $I' = \mathsf{jump}\ \mathsf{r1}$ expects a label in $\mathsf{r1}$ instead of an integer.

Hence the code at l is not well-typed.

Solution:

With each instruction sequence, associate a register file type that is expected at the beginning of that instruction sequence.

Secondly, enrich the notion of types. Instead of having a simple type Code for labels, we have types of the form $\mathsf{Code}(\Gamma)$ where Γ is a register file type.

We further choose a type Top which is the super type of all types.

In the previous example, the instruction sequence I' will have type

$$\{\mathsf{r1} : \mathsf{Code}\{\mathsf{r1} : \mathsf{Top}, \mathsf{r2} : \mathsf{Top}\}\}$$

The instruction sequence I' expects $\mathsf{r1}$ to contain label to some instruction sequence (I) which expects both registers to contain "anything".

The instruction sequence I has type $\{\mathsf{r1} : \mathsf{Top}, \mathsf{r2} : \mathsf{Top}\}$.

After executing the first two instructions of I, the register file type becomes $\{r1 : \mathsf{Int}, \mathsf{r2} : \mathsf{Code}\{\ldots\}\}$.

Hence the jump instruction doesn't type check.

The TAL-0 type system

$\tau ::=$	operand types
Int	integers
$\mathsf{Code}(\Gamma)$	labels
Top	"any" type

The TAL-0 type system

$\tau ::=$		operand types	$\Gamma ::=$		register file types
	Int	integers		$\{\mathsf{r1} : \tau_1, \ldots, \mathsf{rk} : \tau_k\}$	
	$\mathsf{Code}(\Gamma)$	labels	$\Psi ::=$		heap types
	Top	"any" type		$\{l_1 : \tau_1, \ldots, l_m : \tau_m\}$	

The TAL-0 type system

$\tau ::=$	operand types	$\Gamma ::=$	register file types
Int	integers		$\{\mathsf{r1} : \tau_1, \ldots, \mathsf{rk} : \tau_k\}$
$\mathsf{Code}(\Gamma)$	labels	$\Psi ::=$	heap types
Top	"any" type		$\{l_1 : \tau_1, \ldots, l_m : \tau_m\}$

Typing of operands

The type judgment

$$\Psi, \Gamma \vdash \nu : \tau$$

means: under heap type Ψ and register file type Γ, the operand ν has type τ.

The TAL-0 type system

$\tau ::=$	operand types	$\Gamma ::=$	register file types
Int	integers		$\{\mathsf{r1} : \tau_1, \ldots, \mathsf{rk} : \tau_k\}$
$\mathsf{Code}(\Gamma)$	labels	$\Psi ::=$	heap types
Top	"any" type		$\{l_1 : \tau_1, \ldots, l_m : \tau_m\}$

Typing of operands

The type judgment

$$\Psi, \Gamma \vdash \nu : \tau$$

means: under heap type Ψ and register file type Γ, the operand ν has type τ.

$$\Psi, \Gamma \vdash n : \mathsf{Int} \qquad \text{(T-Int)}$$

The TAL-0 type system

$\tau ::=$		operand types	$\Gamma ::=$		register file types
	Int	integers		$\{\mathsf{r1} : \tau_1, \ldots, \mathsf{rk} : \tau_k\}$	
	$\mathsf{Code}(\Gamma)$	labels	$\Psi ::=$		heap types
	Top	"any" type		$\{l_1 : \tau_1, \ldots, l_m : \tau_m\}$	

Typing of operands

The type judgment

$$\Psi, \Gamma \vdash \nu : \tau$$

means: under heap type Ψ and register file type Γ, the operand ν has type τ.

$$\Psi, \Gamma \vdash n : \mathsf{Int} \quad \text{(T-Int)} \qquad \frac{l : \tau \in \Psi}{\Psi, \Gamma \vdash l : \tau} \quad \text{(T-Lab)}$$

$$\Psi, \Gamma \vdash r : \Gamma(r) \qquad \text{(T-Reg)}$$

$$\Psi, \Gamma \vdash r : \Gamma(r) \qquad \text{(T-Reg)}$$

$$\frac{\Psi, \Gamma \vdash \nu : \tau \qquad \tau' \sqsubseteq \tau}{\Psi, \Gamma \vdash \nu : \tau'} \ \text{(T-Sub)}$$

$$\Psi, \Gamma \vdash r : \Gamma(r) \qquad \text{(T-Reg)}$$

$$\frac{\Psi, \Gamma \vdash \nu : \tau \qquad \tau' \sqsubseteq \tau}{\Psi, \Gamma \vdash \nu : \tau'} \text{(T-Sub)}$$

where

$$\begin{array}{rll}
\tau & \sqsubseteq_1 \tau & \text{for every } \tau \\
\tau & \sqsubseteq_1 \mathsf{Top} & \text{for every } \tau \\
\mathsf{Code}(\Gamma_1) & \sqsubseteq \mathsf{Code}(\Gamma_2) & \text{iff } \Gamma_1(r) \sqsubseteq_1 \Gamma_2(r) \text{ for every register } r
\end{array}$$

Top represents "any" type, hence can be replaced by any type.

Typing of instructions

The type judgment

$$\Psi \vdash \iota : \Gamma_1 \rightarrow \Gamma_2$$

means: under heap type Ψ, the instruction ι modifies the register file type from Γ_1 to Γ_2.

Typing of instructions

The type judgment

$$\Psi \vdash \iota : \Gamma_1 \to \Gamma_2$$

means: under heap type Ψ, the instruction ι modifies the register file type from Γ_1 to Γ_2.

$$\frac{\Psi, \Gamma \vdash \nu : \tau}{\Psi \vdash r_d := \nu : \Gamma \to \Gamma \oplus \{r_d : \tau\}} \text{(T-Mov)}$$

Typing of instructions

The type judgment

$$\Psi \vdash \iota : \Gamma_1 \to \Gamma_2$$

means: under heap type Ψ, the instruction ι modifies the register file type from Γ_1 to Γ_2.

$$\frac{\Psi, \Gamma \vdash \nu : \tau}{\Psi \vdash r_d := \nu : \Gamma \to \Gamma \oplus \{r_d : \tau\}} \text{(T-Mov)}$$

$$\frac{\Psi, \Gamma \vdash r_s : \mathsf{Int} \qquad \Psi, \Gamma \vdash \nu : \mathsf{Int}}{\Psi \vdash r_d := r_s + \nu : \Gamma \to \Gamma \oplus \{r_d : \mathsf{Int}\}} \text{(T-Add)}$$

The mov and add instructions modify the type of the destination register.

$$\frac{\Psi, \Gamma \vdash r_s : \mathsf{Int} \qquad \Psi, \Gamma \vdash \nu : \mathsf{Code}(\Gamma)}{\Psi \vdash \mathsf{if}\ r_s\ \mathsf{jump}\ \nu : \Gamma \rightarrow \Gamma} \text{(T-If)}$$

Both branches of the if instruction must have the same type.

If the if condition fails then the next instruction is executed with register file of type Γ.

If the if condition succeeds then the jump should be to some instruction sequence which expects register file type Γ.

Typing of instruction sequences

The type judgment

$$\Psi : I : \mathsf{Code}(\Gamma)$$

means: under heap type Ψ, the instruction sequence I expects the register file to have type Γ at the beginning.

Typing of instruction sequences

The type judgment

$$\Psi : I : \mathsf{Code}(\Gamma)$$

means: under heap type Ψ, the instruction sequence I expects the register file to have type Γ at the beginning.

$$\frac{\Psi, \Gamma \vdash \nu : \mathsf{Code}(\Gamma)}{\Psi \vdash \ \mathsf{jump}\ \nu : \mathsf{Code}(\Gamma)}\ \text{(T-Jump)}$$

Typing of instruction sequences

The type judgment

$$\Psi : I : \mathsf{Code}(\Gamma)$$

means: under heap type Ψ, the instruction sequence I expects the register file to have type Γ at the beginning.

$$\frac{\Psi, \Gamma \vdash \nu : \mathsf{Code}(\Gamma)}{\Psi \vdash \ \mathsf{jump}\ \nu : \mathsf{Code}(\Gamma)} \text{(T-Jump)}$$

$$\frac{\Psi \vdash \iota : \Gamma_1 \longrightarrow \Gamma_2 \qquad \Psi \vdash I : \mathsf{Code}(\Gamma_2)}{\Psi \vdash \iota; I : \mathsf{Code}(\Gamma_1)} \text{(T-Seq)}$$

Typing of register files, heaps, and machine states

$$\frac{\Psi, \_ \vdash R(\mathsf{r1}) : \Gamma(\mathsf{r1}) \qquad \ldots \qquad \Psi, \_ \vdash R(\mathsf{rk}) : \Gamma(\mathsf{rk})}{\Psi \vdash R : \Gamma} \text{(T-Regfile)}$$

_ means that the register file type is irrelevant here

Typing of register files, heaps, and machine states

$$\frac{\Psi, \_ \vdash R(\mathsf{r1}) : \Gamma(\mathsf{r1}) \qquad \ldots \qquad \Psi, \_ \vdash R(\mathsf{rk}) : \Gamma(\mathsf{rk})}{\Psi \vdash R : \Gamma}\ \text{(T-Regfile)}$$

$\_$ means that the register file type is irrelevant here

$$\frac{\forall l \in dom(\Psi) \cdot \Psi \vdash H(l) : \Psi(l)}{\vdash H : \Psi}\ \text{(T-Heap)}$$

$dom(\Psi)$ is the set of labels in the domain of Ψ

Typing of register files, heaps, and machine states

$$\frac{\Psi, \_ \vdash R(\mathsf{r1}) : \Gamma(\mathsf{r1}) \qquad \ldots \qquad \Psi, \_ \vdash R(\mathsf{rk}) : \Gamma(\mathsf{rk})}{\Psi \vdash R : \Gamma}\ \text{(T-Regfile)}$$

_ means that the register file type is irrelevant here

$$\frac{\forall l \in dom(\Psi) \cdot \Psi \vdash H(l) : \Psi(l)}{\vdash H : \Psi}\ \text{(T-Heap)}$$

$dom(\Psi)$ is the set of labels in the domain of Ψ

$$\frac{\vdash H : \Psi \qquad \Psi \vdash R : \Gamma \qquad \Psi \vdash I : \mathsf{Code}(\Gamma)}{\vdash (H, R, I)}\ \text{(T-Mach)}$$

The last judgment means that (H, R, I) is a well-typed machine.

Example

$$\mathsf{l} : \underbrace{\mathsf{r1} := \mathsf{l}; \mathsf{r2} := \mathsf{l}';\ \mathsf{jump\ r2}}_{I} \qquad\qquad \mathsf{l}' : \underbrace{\ \mathsf{jump\ r1}}_{I'}$$

We have the heap $H = \{\mathsf{l} \mapsto I, \mathsf{l}' \mapsto I'\}$.

Define heap type $\Psi = \left\{ \begin{array}{l} \mathsf{l} : \mathsf{Code}\{\mathsf{r1} : \mathsf{Top}, \mathsf{r2} : \mathsf{Top}\}, \\ \mathsf{l}' : \mathsf{Code}\{\mathsf{r1} : \Psi(\mathsf{l}), \mathsf{r2} : \mathsf{Top}\} \end{array} \right\}$

Define register file types
$$\begin{aligned} \Gamma_1 &= \{\mathsf{r1} : \mathsf{Top}, \mathsf{r2} : \mathsf{Top}\} \\ \Gamma_2 &= \{\mathsf{r1} : \Psi(\mathsf{l}), \mathsf{r2} : \mathsf{Top}\} \\ \Gamma_3 &= \{\mathsf{r1} : \Psi(\mathsf{l}), \mathsf{r2} : \Psi(\mathsf{l}')\} \end{aligned}$$

$\boxed{\text{claim 1: } \Psi \vdash I : \mathsf{Code}(\Gamma_1)}$

$\boxed{\text{claim 1: } \Psi \vdash I : \mathsf{Code}(\Gamma_1)}$

$$\dfrac{\dfrac{\mathsf{l} : \mathsf{Code}\{\mathsf{r1} : \mathsf{Top}, \mathsf{r2} : \mathsf{Top}\} \in \Psi}{\Psi, \Gamma_1 \vdash \mathsf{l} : \Psi(\mathsf{l})}\,(\text{T-Lab})}{\Psi \vdash \mathsf{r1} := \mathsf{l} : \Gamma_1 \to \Gamma_2}\,(\text{T-Mov})$$

$$\boxed{\text{claim 1: } \Psi \vdash I : \mathsf{Code}(\Gamma_1)}$$

$$\dfrac{\dfrac{\mathsf{l} : \mathsf{Code}\{\mathsf{r1} : \mathsf{Top}, \mathsf{r2} : \mathsf{Top}\} \in \Psi}{\Psi, \Gamma_1 \vdash \mathsf{l} : \Psi(\mathsf{l})}\,(\text{T-Lab})}{\Psi \vdash \mathsf{r1} := \mathsf{l} : \Gamma_1 \to \Gamma_2}\,(\text{T-Mov}) \qquad \begin{array}{c} \vdots \\ \Psi \vdash \mathsf{r2} := \mathsf{l}' : \Gamma_2 \to \Gamma_3 \end{array}$$

$\boxed{\text{claim 1: } \Psi \vdash I : \mathsf{Code}(\Gamma_1)}$

$$\dfrac{\dfrac{\mathsf{l} : \mathsf{Code}\{\mathsf{r1} : \mathsf{Top}, \mathsf{r2} : \mathsf{Top}\} \in \Psi}{\Psi, \Gamma_1 \vdash \mathsf{l} : \Psi(\mathsf{l})}\ (\text{T-Lab})}{\Psi \vdash \mathsf{r1} := \mathsf{l} \ : \Gamma_1 \to \Gamma_2}\ (\text{T-Mov}) \qquad \begin{array}{c} \vdots \\ \Psi \vdash \mathsf{r2} := \mathsf{l}' \ : \Gamma_2 \to \Gamma_3 \end{array}$$

$$\dfrac{\dfrac{\Psi, \Gamma_3 \vdash \mathsf{r2} : \Psi(\mathsf{l}') \quad \mathsf{Code}(\Gamma_3) \sqsubseteq \Psi(\mathsf{l}')}{\Psi, \Gamma_3 \vdash \mathsf{r2} : \mathsf{Code}(\Gamma_3)}\ (\text{T-Sub})}{\Psi \vdash \ \mathsf{jump}\ \mathsf{r2} : \mathsf{Code}(\Gamma_3)}\ (\text{T-Jump})$$

$$\begin{array}{ll} \mathsf{Code}(\Gamma_3) & = \mathsf{Code}\{\mathsf{r1} : \Psi(\mathsf{l}), \quad \mathsf{r2} : \Psi(\mathsf{l}')\} \\ \sqsubseteq \Psi(\mathsf{l}') & = \mathsf{Code}\{\mathsf{r1} : \Psi(\mathsf{l}), \quad \mathsf{r2} : \mathsf{Top}\} \end{array}$$

because $\Psi(\mathsf{l}) \sqsubseteq_1 \Psi(\mathsf{l})$ and $\Psi(\mathsf{l}') \sqsubseteq_1 \mathsf{Top}$.

$$\dfrac{\dfrac{\vdots}{\Psi \vdash \mathsf{r1} := \mathsf{l} : \Gamma_1 \to \Gamma_2} \qquad \dfrac{\dfrac{\vdots}{\Psi : \mathsf{r2} := \mathsf{l}' : \Gamma_2 \to \Gamma_3} \qquad \dfrac{\vdots}{\Psi \vdash \mathsf{jump}\ \mathsf{r2} : \mathsf{Code}(\Gamma_3)}}{\Psi \vdash \mathsf{r2} := \mathsf{l}';\ \mathsf{jump}\ \mathsf{r2} : \mathsf{Code}(\Gamma_2)}\ (\text{T-Seq})}{\Psi \vdash I : \mathsf{Code}(\Gamma_1)}\ (\text{T-Seq})$$

This proves claim 1.

$$\dfrac{\dfrac{\vdots}{\Psi \vdash \mathsf{r1} := \mathsf{l} : \Gamma_1 \rightarrow \Gamma_2} \qquad \dfrac{\dfrac{\vdots}{\Psi : \mathsf{r2} := \mathsf{l}' : \Gamma_2 \rightarrow \Gamma_3} \qquad \dfrac{\vdots}{\Psi \vdash \ \mathsf{jump}\ \mathsf{r2} : \mathsf{Code}(\Gamma_3)}}{\Psi \vdash \mathsf{r2} := \mathsf{l}';\ \mathsf{jump}\ \mathsf{r2} : \mathsf{Code}(\Gamma_2)}\ (\text{T-Seq})}{\Psi \vdash I : \mathsf{Code}(\Gamma_1)}\ (\text{T-Seq})$$

This proves claim 1.

$$\boxed{\text{claim 2: } \Psi \vdash I' : \mathsf{Code}(\Gamma_2)}$$

$$\dfrac{\dfrac{\Psi, \Gamma_2 \vdash \mathsf{r1} : \Psi(\mathsf{l}) \quad \mathsf{Code}(\Gamma_2) \sqsubseteq \Psi(\mathsf{l})}{\Psi, \Gamma_2 \vdash \mathsf{r1} : \mathsf{Code}(\Gamma_2)}\ (\text{T-Sub})}{\Psi \vdash \ \mathsf{jump}\ \mathsf{r1} : \mathsf{Code}(\Gamma_2)}\ (\text{T-Jump})$$

Well typing of the heap

Recall that $H = \{\mathsf{l} \mapsto I, \mathsf{l}' \mapsto I'\}$ and $\Psi = \{\mathsf{l} : \mathsf{Code}(\Gamma_1), \mathsf{l}' : \mathsf{Code}(\Gamma_2)\}$.

$$\dfrac{\begin{array}{c}\vdots\\ \Psi \vdash I : \mathsf{Code}(\Gamma_1)\end{array} \qquad \begin{array}{c}\vdots\\ \Psi \vdash I' : \mathsf{Code}(\Gamma_2)\end{array}}{\vdash H : \Psi}\ \text{(T-Heap)}$$

Well typing of the heap

Recall that $H = \{\mathsf{l} \mapsto I, \mathsf{l}' \mapsto I'\}$ and $\Psi = \{\mathsf{l} : \mathsf{Code}(\Gamma_1), \mathsf{l}' : \mathsf{Code}(\Gamma_2)\}$.

$$\frac{\begin{array}{c}\vdots\\ \Psi \vdash I : \mathsf{Code}(\Gamma_1)\end{array} \qquad \begin{array}{c}\vdots\\ \Psi \vdash I' : \mathsf{Code}(\Gamma_2)\end{array}}{\vdash H : \Psi}\ (\text{T-Heap})$$

Well typing of register file

Suppose we want to start running the machine with the register file

$$R = \{\mathsf{r1} \mapsto \mathsf{0}, \mathsf{r2} \mapsto \mathsf{0}\}$$

Well typing of the heap

Recall that $H = \{\mathsf{l} \mapsto I, \mathsf{l}' \mapsto I'\}$ and $\Psi = \{\mathsf{l} : \mathsf{Code}(\Gamma_1), \mathsf{l}' : \mathsf{Code}(\Gamma_2)\}$.

$$\frac{\begin{array}{c}\vdots \\ \Psi \vdash I : \mathsf{Code}(\Gamma_1)\end{array} \qquad \begin{array}{c}\vdots \\ \Psi \vdash I' : \mathsf{Code}(\Gamma_2)\end{array}}{\vdash H : \Psi}\ \text{(T-Heap)}$$

Well typing of register file

Suppose we want to start running the machine with the register file

$$R = \{\mathsf{r1} \mapsto \mathsf{0}, \mathsf{r2} \mapsto \mathsf{0}\}$$

Define register file type $\quad \Gamma = \{\mathsf{r1} : \mathsf{Int}, \mathsf{r2} : \mathsf{Int}\}$

Well typing of the heap

Recall that $H = \{\mathsf{l} \mapsto I, \mathsf{l}' \mapsto I'\}$ and $\Psi = \{\mathsf{l} : \mathsf{Code}(\Gamma_1), \mathsf{l}' : \mathsf{Code}(\Gamma_2)\}$.

$$\dfrac{\begin{array}{c}\vdots \\ \Psi \vdash I : \mathsf{Code}(\Gamma_1)\end{array} \qquad \begin{array}{c}\vdots \\ \Psi \vdash I' : \mathsf{Code}(\Gamma_2)\end{array}}{\vdash H : \Psi}\ (\text{T-Heap})$$

Well typing of register file

Suppose we want to start running the machine with the register file

$$R = \{\mathsf{r1} \mapsto \mathsf{0}, \mathsf{r2} \mapsto \mathsf{0}\}$$

Define register file type $\quad \Gamma = \{\mathsf{r1} : \mathsf{Int}, \mathsf{r2} : \mathsf{Int}\}$

$$\dfrac{\dfrac{}{\Psi, \_ \vdash \mathsf{0} : \mathsf{Int}}\ (\text{T-Int}) \qquad \dfrac{}{\Psi, \_ \vdash \mathsf{0} : \mathsf{Int}}\ (\text{T-Int})}{\Psi \vdash R : \Gamma}\ (\text{TRegfile})$$

Suppose the initial instruction sequence we want to execute is I.

We have shown that $\Psi \vdash I : \mathsf{Code}(\Gamma_1)$ (claim 1).

Similarly we show $\Psi \vdash I : \mathsf{Code}(\Gamma)$.

Suppose the initial instruction sequence we want to execute is I.

We have shown that $\Psi \vdash I : \mathsf{Code}(\Gamma_1)$ (claim 1).

Similarly we show $\Psi \vdash I : \mathsf{Code}(\Gamma)$.

Finally, well typing of the machine

$$\frac{\begin{array}{c}\vdots\\ \vdash H : \Psi\end{array} \qquad \begin{array}{c}\vdots\\ \Psi \vdash R : \Gamma\end{array} \qquad \begin{array}{c}\vdots\\ \Psi \vdash I : \mathsf{Code}(\Gamma)\end{array}}{\vdash (H, R, I)} \text{(T-Mach)}$$

Following instruction sequences are rejected by our type system.

$\mathsf{l1} : \mathsf{r1} := \mathsf{l2}; \mathsf{r2} := \mathsf{r1} + \mathsf{1}; \ldots$

$\mathsf{l3} : \mathsf{r1} := \mathsf{5};\ \mathsf{jump\ r1}$

Following instruction sequences are rejected by our type system.

$\mathsf{l1} : \mathsf{r1} := \mathsf{l2}; \mathsf{r2} := \mathsf{r1} + \mathsf{1}; \ldots$

$\mathsf{l3} : \mathsf{r1} := \mathsf{5};\ \mathsf{jump}\ \mathsf{r1}$

- We haven't discussed how to check if a machine is well typed. Alternative: use proof carrying code.

Following instruction sequences are rejected by our type system.

$\mathsf{l1} : \mathsf{r1} := \mathsf{l2}; \mathsf{r2} := \mathsf{r1} + \mathsf{1}; \ldots$

$\mathsf{l3} : \mathsf{r1} := \mathsf{5};\ \mathsf{jump}\ \mathsf{r1}$

- We haven't discussed how to check if a machine is well typed. Alternative: use proof carrying code.
- It is straightforward to translate TAL-0 programs to code for some real processor.

  If the TAL-0 program is well-typed then the translated code will behave properly.

Following instruction sequences are rejected by our type system.

$\mathsf{l1} : \mathsf{r1} := \mathsf{l2}; \mathsf{r2} := \mathsf{r1} + \mathsf{1}; \ldots$

$\mathsf{l3} : \mathsf{r1} := \mathsf{5};\ \mathsf{jump}\ \mathsf{r1}$

- We haven't discussed how to check if a machine is well typed. Alternative: use proof carrying code.
- It is straightforward to translate TAL-0 programs to code for some real processor.

  If the TAL-0 program is well-typed then the translated code will behave properly.

  . . . for that we of course need to prove type safety for TAL-0 . . .

Type safety for TAL-0

"well typed machines do not get stuck"

Progress: If $\vdash M$ then there is some M' such that $M \rightarrow M'$.

Preservation: If $\vdash M$ and $M \rightarrow M'$ then $\vdash M'$.