Checks and Balances - Constraint Solving without Surprises in Object-Constraint Programming Languages: Full Formal Development

Tim Felgentreff, Todd Millstein, Alan Borning
and Robert Hirschfeld

VPRI Technical Report TR-2015-001

Viewpoints Research Institute, 1025 Westwood Blvd 2nd flr, Los Angeles, CA 90024 t: (310) 208-0524

Checks and Balances – Constraint Solving without Surprises in Object-Constraint Programming Languages: Full Formal Development

Tim Felgentreff, Todd Millstein, Alan Borning, and Robert Hirschfeld

VPRI Technical Report TR-2015-001, August 2015

1 Babelsberg/UID and Two Key Theorems

This technical report is intended to accompany the conference paper "Checks and Balances – Constraint Solving without Surprises in Object-Constraint Programming Languages," and presents the full formalism and proofs for two theorems that capture key properties of the Babelsberg/UID core language. These theorems are stated (without the proofs) in Section 3.3 of that paper.

1.1 Syntax

We use the following syntax for Babelsberg/UID:

Statement	`s`	::=	`skip` | `L := e` | `x := new o` | `always C` | `once C` | `s;s` | `if e then s else s` | `while e do s`
Constraint	`C`	::=	ρ `e` | `C` $\wedge$ `C`
Expression	`e`	::=	`v` | `L` | `e` $\oplus$ `e` | `L==L` | `D`
Object Literal	`o`	::=	$\{$`l`$_1$`:e`$_1$`,...,l`$_n$`:e`$_n\}$
L-Value	`L`	::=	`x` | `L.l`
Constant	`c`	::=	`true` | `false` | `nil` | base type constants
Variable	`x`	::=	variable names
Label	`l`	::=	record label names
Reference	`r`	::=	references to heap records
Dereference	`D`	::=	$\mathbb{H}$`(e)`
Value	`v`	::=	`c` | `r`

Metavariable `c` ranges over the `nil` value, booleans, and primitive type constants. A finite set of operators on primitives is ranged over by $\oplus$. We assume that $\oplus$ includes an equality operator for each primitive type; for convenience we use the symbol `=` to denote each of these operators. We also assume it includes an operator $\wedge$ for boolean conjunction. The operator `==` tests for identity — for primitive values this behaves the same as `=`. The symbol ρ ranges over constraint *priorities* and is assumed to include a bottom element `weak` and a top element `required`. The syntax requires the priority to be explicit; for simplicity we sometimes omit it in the rules and assume a `required` priority.

In the syntax, we treat `H` as a keyword used for dereferencing. Source programs will not use expressions of the form $\mathbb{H}$`(e)`, but they are introduced as part of constraints given to the solver, which we assume will treat `H` as an uninterpreted function. We also assume that the solver supports records and record equality, which we also denote with the `=` operator.

1.2 Operational Semantics

The semantics includes an environment `E` and a heap `H`. The former is a function that maps variable names to values, while the latter is a function that maps mutable references to "objects" of the form $\{\mathtt{l}_1\mathtt{:v}_1\mathtt{,...,l}_n\mathtt{:v}_n\}$. When convenient, we also treat both `E` and `H` as a set of pairs ($\{$`(x,v),...`$\}$ and $\{$`(r,o),...`$\}$, respectively). The currently active value constraints are kept as a compound constraint $\mathbb{C}$; identity constraints are kept as a single conjunction referred to as $\mathbb{I}$.

$\boxed{\mathtt{E;H} \vdash \mathtt{e} \Downarrow \mathtt{v}}$

"Expression `e` evaluates to value `v` in the context of environment `E` and heap `H`."

The rules for evaluation are mostly as expected in an imperative language. We do not give rules for expressions of the form $\mathbb{H}$`(e)`, because they are not meant to appear in source. For each operator $\oplus$ in the language we assume the existence of a corresponding semantic function denoted $[\![\oplus]\!]$.

$$\mathtt{E;H} \vdash \mathtt{c} \Downarrow \mathtt{c} \qquad \text{(E-Const)}$$

$$\frac{\mathtt{E(x)} = \mathtt{v}}{\mathtt{E;H} \vdash \mathtt{x} \Downarrow \mathtt{v}} \qquad \text{(E-Var)}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{L} \Downarrow \mathtt{r} \qquad \mathtt{H(r)} = \{\mathtt{l}_1\mathtt{:v}_1\mathtt{,...,l}_n\mathtt{:v}_n\} \qquad 1 \le i \le n}{\mathtt{E;H} \vdash \mathtt{L.l}_i \Downarrow \mathtt{v}_i} \qquad \text{(E-Field)}$$

$$\mathtt{E;H} \vdash \mathtt{r} \Downarrow \mathtt{r} \qquad \text{(E-Ref)}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{e}_1 \Downarrow \mathtt{v}_1 \qquad \mathtt{E;H} \vdash \mathtt{e}_2 \Downarrow \mathtt{v}_2 \qquad \mathtt{v}_1\ [\![\oplus]\!]\ \mathtt{v}_2 = \mathtt{v}}{\mathtt{E;H} \vdash \mathtt{e}_1\ \oplus\ \mathtt{e}_2 \Downarrow \mathtt{v}} \qquad \text{(E-Op)}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{L}_1 \Downarrow \mathtt{v} \qquad \mathtt{E;H} \vdash \mathtt{L}_2 \Downarrow \mathtt{v}}{\mathtt{E;H} \vdash \mathtt{L}_1\ \mathtt{==}\ \mathtt{L}_2 \Downarrow \mathtt{true}} \qquad \text{(E-IdentityTrue)}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{L}_1 \Downarrow \mathtt{v}_1 \qquad \mathtt{E;H} \vdash \mathtt{L}_2 \Downarrow \mathtt{v}_2 \qquad \mathtt{v}_1 \neq \mathtt{v}_2}{\mathtt{E;H} \vdash \mathtt{L}_1\ \mathtt{==}\ \mathtt{L}_2 \Downarrow \mathtt{false}} \qquad \text{(E-IdentityFalse)}$$

$\boxed{\mathtt{E;H} \vdash \mathtt{e} : \mathtt{T}}$

$\boxed{\mathtt{E;H} \vdash \mathtt{C}}$

"Expression `e` has type `T` in the context of environment `E` and heap `H`."

"Constraint `C` is well formed in the context of environment `E` and heap `H`."

We use a notion of typechecking to prevent undesirable non-determinism in constraints. Specifically, we want constraint solving to preserve the structure of the values of variables, changing only the underlying primitive data as part of a solution. We formalize our notion of structure through a simple syntax of types:

$$\text{Type} \quad \mathtt{T} \quad ::= \quad \mathtt{PrimitiveType} \mid \{\mathtt{l}_1{:}\mathtt{T}_1, \ldots, \mathtt{l}_n{:}\mathtt{T}_n\}$$

The typechecking rules are mostly standard. We check expressions dynamically just before constraint solving, so we typecheck in the context of a runtime environment. Note that we do not include type rules for identities. This ensures that constraints involving them do not typecheck, so identity checks cannot occur in ordinary constraints.

$$\mathtt{E;H} \vdash \mathtt{c} : \mathtt{PrimitiveType} \tag{T-Const}$$

$$\frac{\mathtt{H(r)}=\{\mathtt{l}_1{:}\mathtt{v}_1, \ldots, \mathtt{l}_n{:}\mathtt{v}_n\} \qquad \mathtt{E;H} \vdash \mathtt{v}_1 : \mathtt{T}_1 \cdots \mathtt{E;H} \vdash \mathtt{v}_n : \mathtt{T}_n}{\mathtt{E;H} \vdash \mathtt{r} : \{\mathtt{l}_1{:}\mathtt{T}_1, \ldots, \mathtt{l}_n{:}\mathtt{T}_n\}} \tag{T-Ref}$$

$$\frac{\mathtt{E(x)} = \mathtt{v} \qquad \mathtt{E;H} \vdash \mathtt{v} : \mathtt{T}}{\mathtt{E;H} \vdash \mathtt{x} : \mathtt{T}} \tag{T-Var}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{L} : \{\mathtt{l}_1{:}\mathtt{T}_1, \ldots, \mathtt{l}_n{:}\mathtt{T}_n\} \qquad 1 \leq i \leq n}{\mathtt{E;H} \vdash \mathtt{L.l}_i : \mathtt{T}_i} \tag{T-Field}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{e}_1 : \mathtt{PrimitiveType} \qquad \mathtt{E;H} \vdash \mathtt{e}_2 : \mathtt{PrimitiveType}}{\mathtt{E;H} \vdash \mathtt{e}_1 \ \oplus \ \mathtt{e}_2 : \mathtt{PrimitiveType}} \tag{T-Op}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{e} : \mathtt{T}}{\mathtt{E;H} \vdash \rho \ \mathtt{e}} \tag{T-Priority}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{C}_1 \qquad \mathtt{E;H} \vdash \mathtt{C}_2}{\mathtt{E;H} \vdash \mathtt{C}_1 \ \wedge \ \mathtt{C}_2} \tag{T-Conjunction}$$

$\boxed{\mathtt{E;H} \models \mathtt{C}}$

This judgment represents a call to the constraint solver, which we treat as a black box. The proposition $\mathtt{E;H} \models \mathbb{C}$ denotes that environment `E` and heap `H` are an *optimal solution* to the constraint `C`, according to the solver's semantics.

We assume several well-formedness properties about a solution `E` and `H` to constraints `C`:

- any object reference appearing in the range of `E` also appears in the domain of `H`
- any object reference appearing in the range of `H` also appears in the domain of `H`
- for all variables `x` in the domain of `E` there is some type `T` such that $\mathtt{E;H} \vdash \mathtt{x} : \mathtt{T}$
- $\mathtt{E;H} \vdash \mathtt{C}$

$\boxed{\text{stay}(\mathtt{x=v}, \rho) = \mathtt{C}}$

$\boxed{\text{stay}(\texttt{r=o},\ \rho) = \texttt{C}}$

$\boxed{\text{stay}(\texttt{E},\ \rho) = \texttt{C}}$

$\boxed{\text{stay}(\texttt{H},\ \rho) = \texttt{C}}$

As in Kaleidoscope, the semantics ensure that each variable has a stay constraint to keep it at its current value, if possible. The stay rules take a priority as a parameter. When solving value constraints, this priority is set to `required`, to ensure that the structures of objects and the relationship between l-values and object references cannot change. When solving identity constraints as part of executing an assignment statement, the priority is set to `weak` to allow structural changes.

To properly account for the heap in the constraint solver, we employ an uninterpreted function `H` that maps references to objects (i.e., records). The rules below employ this function in order to define stay constraints for references.

$$\text{stay}(\texttt{x=c},\ \rho) = \texttt{weak x=c} \qquad \text{(StayConst)}$$

$$\text{stay}(\texttt{x=r},\ \rho) = \rho\ \texttt{x=r} \qquad \text{(StayRef)}$$

$$\frac{\texttt{x}_1 \text{ fresh} \cdots \texttt{x}_n \text{ fresh} \qquad \text{stay}(\texttt{x}_1\texttt{=v}_1,\ \rho) = \texttt{C}_1 \cdots \text{stay}(\texttt{x}_n\texttt{=v}_n,\ \rho) = \texttt{C}_n}{\text{stay}(\texttt{r = \{l}_1\texttt{:v}_1\texttt{,...,l}_n\texttt{:v}_n\texttt{}\},\ \rho) = \texttt{(required } \mathbb{H}\texttt{(r)=\{l}_1\texttt{:x}_1\texttt{,...,l}_n\texttt{:x}_n\texttt{\})} \wedge \texttt{C}_1 \wedge \cdots \wedge \texttt{C}_n} \qquad \text{(StayObject)}$$

$$\frac{\texttt{E= \{(x}_1\texttt{, v}_1\texttt{), ..., (x}_n\texttt{, v}_n\texttt{)\}} \qquad \text{stay}(\texttt{x}_1\texttt{=v}_1,\ \rho) = \texttt{C}_1 \cdots \text{stay}(\texttt{x}_n\texttt{=v}_n,\ \rho) = \texttt{C}_n}{\text{stay}(\texttt{E},\ \rho) = \texttt{C}_1 \wedge \cdots \wedge \texttt{C}_n} \qquad \text{(StayEnv)}$$

$$\frac{\texttt{H= \{(r}_1\texttt{, o}_1\texttt{), ..., (r}_n\texttt{, o}_n\texttt{)\}} \qquad \text{stay}(\texttt{r}_1\texttt{=o}_1,\ \rho) = \texttt{C}_1 \cdots \text{stay}(\texttt{r}_n\texttt{=o}_n,\ \rho) = \texttt{C}_n}{\text{stay}(\texttt{H},\ \rho) = \texttt{C}_1 \wedge \cdots \wedge \texttt{C}_n} \qquad \text{(StayHeap)}$$

$\boxed{\text{stayPrefix}(\texttt{E},\ \texttt{H},\ \texttt{L}) = \texttt{C}}$

$\boxed{\text{stayPrefix}(\texttt{E},\ \texttt{H},\ \texttt{I}) = \texttt{C}}$

These judgments are another form of stay constraints that ensure that the "prefix" `L` of an l-value `L.l` is unchanged; this is necessary to ensure that updates to the value of `L.l` are deterministic.

$$\text{stayPrefix}(\texttt{E},\ \texttt{H},\ \texttt{x}) = \texttt{true} \qquad \text{(StayPrefixVar)}$$

$$\frac{\texttt{L} = \texttt{x.l}_1\texttt{.....l}_n \qquad n > 0 \qquad \texttt{E;H} \vdash \texttt{x} \Downarrow \texttt{r}_0 \qquad \texttt{E;H} \vdash \texttt{r}_0\texttt{.l}_1 \Downarrow \texttt{r}_1 \cdots \texttt{E;H} \vdash \texttt{r}_{n-2}\texttt{.l}_{n-1} \Downarrow \texttt{r}_{n-1}}{\text{stayPrefix}(\texttt{E},\ \texttt{H},\ \texttt{L}) = \texttt{x=r}_0 \wedge \texttt{r}_0\texttt{.l}_1\texttt{=r}_1 \wedge \cdots \wedge \texttt{r}_{n-2}\texttt{.l}_{n-1}\texttt{=r}_{n-1}} \qquad \text{(StayPrefixField)}$$

$$\frac{\texttt{I} = \texttt{L}_1\texttt{==L}_2 \wedge \cdots \wedge \texttt{L}_{2n-1}\texttt{==L}_{2n} \qquad \text{stayPrefix}(\texttt{E},\ \texttt{H},\ \texttt{L}_1) = \texttt{C}_1 \cdots \text{stayPrefix}(\texttt{E},\ \texttt{H},\ \texttt{L}_{2n}) = \texttt{C}_{2n}}{\text{stayPrefix}(\texttt{E},\ \texttt{H},\ \texttt{I}) = \texttt{C}_1 \wedge \cdots \wedge \texttt{C}_{2n}} \qquad \text{(StayPrefixIdent)}$$

$\boxed{\texttt{E;H} \vdash \texttt{C} \rightsquigarrow \texttt{C}'}$

We use these judgments to translate a constraint into a constraint suitable for the solver. Specifically, each occurrence of an expression of the form `L.l`, where `L` refers to a heap reference `r`, is translated into $\mathbb{H}$`(L).l` (recursively, as required), and each occurrence of the identity operator `==` is replaced by ordinary equality. We do not give these rules, because they are straightforward.

$\boxed{\text{solve}(\texttt{E}, \texttt{H}, \mathbb{C}, \rho) = \texttt{E}';\texttt{H}'}$

"Solving constraint $\mathbb{C}$ in the context of `E` and `H` using stay constraints with priority ρ produces the new environment and heap `E`′ and `H`′."

This judgment represents one phase of constraint solving – either solving "value" constraints or identity constraints.

$$\frac{\text{stay}(\texttt{E}, \rho) = \texttt{C}_E \qquad \text{stay}(\texttt{H}, \rho) = \texttt{C}_H \qquad \texttt{E;H} \vdash \mathbb{C} \rightsquigarrow \texttt{C}' \qquad \texttt{E}';\texttt{H}' \models (\texttt{C}' \wedge \texttt{C}_E \wedge \texttt{C}_H)}{\text{solve}(\texttt{E}, \texttt{H}, \mathbb{C}, \rho) = \texttt{E}';\texttt{H}'} \quad \text{(SOLVE)}$$

$\boxed{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|s>} \longrightarrow \texttt{<E}'\texttt{|H}'\texttt{|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}}$

"Execution starting from configuration `<E|H|`$\mathbb{C}$`|`$\mathbb{I}$`|s>` ends in state `<E`′`|H`′`|`$\mathbb{C}$′`|`$\mathbb{I}$′`>`."

A "configuration" defining the state of an execution includes a concrete context, represented by the environment and heap, a symbolic context, represented by the constraint and identity constraint stores, and a statement to be executed. The environment, heap, and statement are standard, while the constraint stores are not part of the state of a computation in most languages. Intuitively, the environment and heap come from constraint solving during the evaluation of the immediately preceding statement, and the constraint records the `always` constraints that have been declared so far during execution. Note that our execution implicitly gets stuck if the solver cannot produce a model.

The rule below describes the semantics of assignments. We employ a two-phase process. First the identity constraints are solved in the context of the new assignment. This phase propagates the effect of the assignment through the identities, potentially changing the structures of objects as well as the relationships among objects in the environment and heap. In the second phase, the value constraints are typechecked against the new environment and heap resulting from the first phase. If they are well typed, then we proceed to solve them. This phase can change the values of primitives but will not modify the structure of any object.

Implicitly this rule gets stuck if either a) the identity constraints cannot be solved, b) the value constraints do not typecheck, or c) the value constraints cannot be solved. A practical implementation would add explicit exceptions for these cases that the programmer could handle.

$$\frac{\texttt{E;H} \vdash \texttt{e} \Downarrow \texttt{v} \qquad \text{stayPrefix}(\texttt{E}, \texttt{H}, \texttt{L}) = \texttt{C}_L \qquad \text{stayPrefix}(\texttt{E}, \texttt{H}, \texttt{I}) = \texttt{C}_I \qquad \text{solve}(\texttt{E}, \texttt{H}, \texttt{C}_L \wedge \texttt{C}_I \wedge \texttt{L=v} \wedge \texttt{I}, \texttt{weak}) = \texttt{E}';\texttt{H}' \qquad \texttt{E}';\texttt{H}' \vdash \texttt{C} \qquad \text{solve}(\texttt{E}', \texttt{H}', \mathbb{C} \wedge \texttt{L=v}, \texttt{required}) = \texttt{E}'';\texttt{H}''}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|L := e>} \longrightarrow \texttt{<E}''\texttt{|H}''\texttt{|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>}} \quad \text{(S-ASGN)}$$

The next rule describes the semantics of object creation, which is straightforward. For simplicity we require a new object to be initially assigned to a fresh variable, but this is no loss of expressiveness.

$$\frac{\mathtt{E}(\mathtt{x})\uparrow \qquad \mathbb{H}(\mathtt{r})\uparrow \qquad \mathtt{E;H} \vdash \mathtt{e}_1 \Downarrow \mathtt{v}_n \cdots \mathtt{E;H} \vdash \mathtt{e}_n \Downarrow \mathtt{v}_n \qquad \mathtt{E}' = \mathtt{E} \bigcup \{(\mathtt{x}, \mathtt{r})\} \qquad \mathtt{H}' = \mathtt{H} \bigcup \{(\mathtt{r}, \{\mathtt{l}_1{:}\mathtt{v}_1, \ldots, \mathtt{l}_n{:}\mathtt{v}_n\})\}}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|x := new } \{\mathtt{l}_1{:}\mathtt{e}_1, \ldots, \mathtt{l}_n{:}\mathtt{e}_n\}\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>}} \text{ (S-AsgnNew)}$$

The next two rules describe the semantics of identity constraints. The rules require than identity constraint already be satisfied when it is asserted; hence the environment and heap are unchanged.

$$\frac{\mathtt{E;H} \vdash \mathtt{L}_0 \Downarrow \mathtt{v} \qquad \mathtt{E;H} \vdash \mathtt{L}_1 \Downarrow \mathtt{v}}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|once } \mathtt{L}_0 \texttt{ == } \mathtt{L}_1\texttt{> } \longrightarrow \texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>}} \text{ (S-OnceIdentity)}$$

$$\frac{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|once } \mathtt{L}_0 \texttt{ == } \mathtt{L}_1\texttt{> } \longrightarrow \texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>} \qquad \mathbb{I}' = \mathbb{I} \wedge \mathtt{L}_0 \texttt{ == } \mathtt{L}_1}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|always } \mathtt{L}_0 \texttt{ == } \mathtt{L}_1\texttt{> } \longrightarrow \texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}'\texttt{>}} \text{ (S-AlwaysIdentity)}$$

The following two rules describe the semantics of value constraints. Recall that these constraints cannot contain identity constraints in them (because identity constraints do not typecheck). As we show later, solving value constraints cannot change the structure of any objects on the environment and heap.

$$\frac{\mathtt{E;H} \vdash \mathtt{C}_0 \qquad \text{solve}(\mathtt{E}, \mathtt{H}, \mathbb{C} \wedge \mathtt{C}_0, \texttt{required}) = \mathtt{E}'\texttt{;}\mathtt{H}'}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|once } \mathtt{C}_0\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>}} \text{ (S-Once)}$$

$$\frac{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|once } \mathtt{C}_0\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>} \qquad \mathbb{C}' = \mathbb{C} \wedge \mathtt{C}_0}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|always } \mathtt{C}_0\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}\texttt{>}} \text{ (S-Always)}$$

The remaining rules are standard for imperative languages, only augmented with constraint stores, and are only given for completeness.

$$\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|skip> } \longrightarrow \texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>} \text{ (S-Skip)}$$

$$\frac{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|}\mathtt{s}_1\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>} \qquad \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{|}\mathtt{s}_2\texttt{> } \longrightarrow \texttt{<E''|H''|}\mathbb{C}''\texttt{|}\mathbb{I}''\texttt{>}}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|}\mathtt{s}_1\texttt{;}\mathtt{s}_2\texttt{> } \longrightarrow \texttt{<E''|H''|}\mathbb{C}''\texttt{|}\mathbb{I}''\texttt{>}} \text{ (S-Seq)}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{e} \Downarrow \texttt{true} \qquad \texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|}\mathtt{s}_1\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|if e then } \mathtt{s}_1 \texttt{ else } \mathtt{s}_2\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}} \text{ (S-IfThen)}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{e} \Downarrow \texttt{false} \qquad \texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|}\mathtt{s}_2\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|if e then } \mathtt{s}_1 \texttt{ else } \mathtt{s}_2\texttt{> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}} \text{ (S-IfElse)}$$

$$\frac{\begin{array}{c}\mathtt{E;H} \vdash \mathtt{e} \Downarrow \texttt{true} \qquad \texttt{<E|}\mathbb{C}\texttt{|H|}\mathbb{I}\texttt{|s> } \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>} \\ \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{|while e do s> } \longrightarrow \texttt{<E''|H''|}\mathbb{C}''\texttt{|}\mathbb{I}''\texttt{>}\end{array}}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|while e do s> } \longrightarrow \texttt{<E''|H''|}\mathbb{C}''\texttt{|}\mathbb{I}''\texttt{>}} \text{ (S-WhileDo)}$$

$$\frac{\mathtt{E;H} \vdash \mathtt{e} \Downarrow \texttt{false}}{\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|while e do s> } \longrightarrow \texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>}} \text{ (S-WhileSkip)}$$

1.3 Properties

Here we state and prove two key theorems about our formalism.

We assume that a given configuration $\texttt{E|H|C|I}$ is *well formed*, meaning that it satisfies these sanity conditions:

- any object reference appearing in the range of $\texttt{E}$ also appears in the domain of $\texttt{H}$
- any object reference appearing in the range of $\texttt{H}$ also appears in the domain of $\texttt{H}$
- for all variables $\texttt{x}$ in the domain of $\texttt{E}$ there is some type $\texttt{T}$ such that $\texttt{E;H} \vdash \texttt{x} : \texttt{T}$
- $\texttt{E;H} \vdash \texttt{C}$
- $\texttt{E;H} \vdash \texttt{I} \Downarrow \texttt{true}$

Well-formedness follows from the assumptions made on the constraint solver described earlier.

The first theorem formalizes the idea that any solution to a value constraint preserves the structures of the objects on the environment and heap:

Theorem 1 *(Structure Preservation) If* $\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|s>} \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}$ *and* $\texttt{s}$ *either has the form* $\texttt{once}\ \texttt{C}_0$ *or* $\texttt{always}\ \texttt{C}_0$ *and* $\texttt{E;H} \vdash \texttt{C}_0$ *and* $\texttt{E;H} \vdash \texttt{L} : \texttt{T}$*, then* $\texttt{E';H'} \vdash \texttt{L} : \texttt{T}$*.*

Proof. If $\texttt{s}$ has the form $\texttt{once}\ \texttt{C}_0$ then the result follows by Lemma 1. If $\texttt{s}$ has the form $\texttt{always}\ \texttt{C}_0$, then since $\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|s>} \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}$ and $\texttt{C}_0$ is not an identity test, by rule S-Always we have $\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|once C}_0\texttt{>} \longrightarrow \texttt{<E'|H'|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{>}$. Then again the result follows from Lemma 1. □

Lemma 1 *If* $\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|once C}_0\texttt{>} \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}$ *and* $\texttt{E;H} \vdash \texttt{C}_0$ *and* $\texttt{E;H} \vdash \texttt{L} : \texttt{T}$*, then* $\texttt{E';H'} \vdash \texttt{L} : \texttt{T}$*.*

Proof. Since $\texttt{<E|H|}\mathbb{C}\texttt{|}\mathbb{I}\texttt{|once C}_0\texttt{>} \longrightarrow \texttt{<E'|H'|}\mathbb{C}'\texttt{|}\mathbb{I}'\texttt{>}$ and $\texttt{C}_0$ is not an identity test, by S-Once we have that $\texttt{E;H} \vdash \texttt{C}_0$ and solve($\texttt{E}$, $\texttt{H}$, $\mathbb{C} \wedge \texttt{C}_0$, $\texttt{required}$) = $\texttt{E';H'}$. By the assumption of well-formedness we have $\texttt{E;H} \vdash \texttt{C}$ so by T-Conjunction also $\texttt{E;H} \vdash \texttt{C} \wedge \texttt{C}_0$. Therefore the result follows from Lemma 2. □

Lemma 2 *If* $\texttt{E;H} \vdash \texttt{C}$ *and* solve($\texttt{E}$, $\texttt{H}$, $\mathbb{C}$, $\texttt{required}$) = $\texttt{E';H'}$ *and* $\texttt{E;H} \vdash \texttt{L} : \texttt{T}$*, then* $\texttt{E';H'} \vdash \texttt{L} : \texttt{T}$*.*

Proof. Since solve($\texttt{E}$, $\texttt{H}$, $\mathbb{C}$, $\texttt{required}$) = $\texttt{E';H'}$ by Solve we have that stay($\texttt{E}$, $\texttt{required}$) = $\texttt{C}_{E_s}$ and stay($\texttt{H}$, $\texttt{required}$) = $\texttt{C}_{H_s}$ and $\texttt{E;H} \vdash \mathbb{C} \rightsquigarrow \texttt{C}'$ and $\texttt{E';H'} \models (\texttt{C}' \wedge \texttt{C}_{E_s} \wedge \texttt{C}_{H_s})$. We prove this lemma by structural induction on $\texttt{T}$.

By Lemma 3 there exists a value $\texttt{v}$ such that $\texttt{E;H} \vdash \texttt{L} \Downarrow \texttt{v}$ and $\texttt{E;H} \vdash \texttt{v} : \texttt{T}$. Then by Lemma 4 there exists a value $\texttt{v}'$ such that $\texttt{E';H'} \vdash \texttt{L} \Downarrow \texttt{v}'$. Furthermore, if $\texttt{v}$ is an object reference $\texttt{r}$, then also $\texttt{v}' = \texttt{r}$. Case analysis on the form of $\texttt{v}'$:

- Case $\texttt{v}'$ is a constant $\texttt{c}'$. Then also $\texttt{v}$ is a constant $\texttt{c}$. Since $\texttt{E;H} \vdash \texttt{v} : \texttt{T}$, the last rule in this derivation must be T-Const so $\texttt{T}$ is $\texttt{PrimitiveType}$ and the result follows from T-Const.
- Case $\texttt{v}'$ is a reference $\texttt{r}'$. Since $\texttt{E';H'} \vdash \texttt{L} \Downarrow \texttt{v}'$ by Lemma 5 there is a type $\texttt{T}'$ such that $\texttt{E';H'} \vdash \texttt{L} : \texttt{T}'$ and $\texttt{E';H'} \vdash \texttt{r}' : \texttt{T}'$. So it suffices to show that $\texttt{T} = \texttt{T}'$. Case analysis on the form of $\texttt{v}$:

– Case $\mathtt{v}$ is a constant $\mathtt{c}$. Since $\mathtt{E;H} \vdash \mathtt{v} : \mathtt{T}$, the last rule in this derivation must be T-Const, so we have $\mathtt{T} = \mathtt{PrimitiveType}$. Case analysis on the last rule in the derivation of $\mathtt{E;H} \vdash \mathtt{L} \Downarrow \mathtt{v}$:

  * Case E-Variable. Then $\mathtt{L}$ is a variable $\mathtt{x}$ and $\mathtt{E}(\mathtt{x}) = \mathtt{c}$. Since stay($\mathtt{E}$, $\mathtt{required}$) $= \mathtt{C}_{E_s}$, by StayEnv we have that $\mathtt{C}_{E_s}$ includes a conjunct $\mathtt{C}_x$ such that stay($\mathtt{x{=}c}$, $\mathtt{required}$) $= \mathtt{C}_x$, and by StayConst $\mathtt{C}_x$ has the form $\mathtt{weak\ x{=}c}$.
    Since $\mathtt{E'};\mathtt{H'} \vdash \mathtt{L} \Downarrow \mathtt{r'}$, the last rule in this derivation must be E-Variable, so we have $\mathtt{E'}(\mathtt{x}) = \mathtt{r'}$. We know that assigning $\mathtt{x}$ to $\mathtt{c}$ satisfies the weak stay constraint above. Therefore there must be some constraint on $\mathtt{x}$ in $\mathtt{C}$ that causes the change in value for $\mathtt{x}$ from $\mathtt{c}$ to $\mathtt{r'}$.
  * Case E-Field. Then $\mathtt{L}$ has the form $\mathtt{L_0.l}_i$ and $\mathtt{E;H} \vdash \mathtt{L_0} \Downarrow \mathtt{r}_0$ and $\mathtt{H}(\mathtt{r}_0) = \{\mathtt{l}_1{:}\mathtt{v}_1, \ldots, \mathtt{l}_n{:}\mathtt{v}_n\}$ and $1 \leq i \leq n$ and $\mathtt{r'} = \mathtt{v}_i$. Since stay($\mathtt{H}$, $\mathtt{required}$) $= \mathtt{C}_{H_s}$, by StayHeap we have $\mathtt{C}_{H_s}$ includes a conjunct $\mathtt{C}_{r_0}$ such that stay($\mathtt{r}_0{=}\{\mathtt{l}_1{:}\mathtt{v}_1, \ldots, \mathtt{l}_n{:}\mathtt{v}_n\}$, $\mathtt{required}$) $= \mathtt{C}_{r_0}$. By StayObject $\mathtt{C}_{r_0}$ has the form $(\mathtt{required}\ \mathbb{H}(\mathtt{r}_0){=}\{\mathtt{l}_1{:}\mathtt{x}_1, \ldots, \mathtt{l}_n{:}\mathtt{x}_n\}) \wedge \mathtt{C}_1 \wedge \cdots \wedge \mathtt{C}_n$, where $\mathtt{x}_1 \ldots \mathtt{x}_n$ are fresh variables and stay($\mathtt{x}_i{=}\mathtt{v}_i$, $\mathtt{required}$) $= \mathtt{C}_i$. Then by StayConst $\mathtt{C}_i$ has the form $\mathtt{weak\ x}_i{=}\mathtt{c}$.
    Since $\mathtt{E;H} \vdash \mathtt{L_0} \Downarrow \mathtt{r}_0$, by Lemma 4 we have $\mathtt{E';H'} \vdash \mathtt{L_0} \Downarrow \mathtt{r}_0$. Since $\mathtt{E';H'} \vdash \mathtt{L} \Downarrow \mathtt{r'}$, the last rule in this derivation must be E-Field, so we have that $\mathtt{H'}(\mathtt{r}_0) = \{\mathtt{l}_1{:}\mathtt{v}'_1, \ldots, \mathtt{l}_n{:}\mathtt{v}'_n\}$ where $\mathtt{v}'_i$, and hence $\mathtt{x}_i$ is $\mathtt{r'}$. We know that assigning $\mathtt{x}_i$ to $\mathtt{c}$ satisfies the weak stay constraint above. Therefore there must be some constraint on an l-value of the form $\mathtt{L}_{00}\mathtt{.l}_i$ in $\mathtt{C}$, where $\mathtt{E;H} \vdash \mathtt{L}_{00} \Downarrow \mathtt{r}_0$, that causes the change in value for $\mathtt{x}_i$ from $\mathtt{c}$ to $\mathtt{r'}$.

  Therefore in either case, there must be some constraint on an l-value $\mathtt{L''}$ in $\mathtt{C}$, where $\mathtt{E;H} \vdash \mathtt{L''} \Downarrow \mathtt{c}$ but $\mathtt{E';H'} \vdash \mathtt{L''} \Downarrow \mathtt{r'}$. By Lemma 5 there is some $\mathtt{T''}$ such that $\mathtt{E;H} \vdash \mathtt{L''} : \mathtt{T''}$ and $\mathtt{E;H} \vdash \mathtt{c} : \mathtt{T''}$. The last rule in the derivation of $\mathtt{E;H} \vdash \mathtt{c} : \mathtt{T''}$ must be T-Const so $\mathtt{T''}$ is $\mathtt{PrimitiveType}$. We argue a contradiction by case analysis of the immediate parent expression of any occurrence of $\mathtt{L''}$ in $\mathtt{C}$. We are given $\mathtt{E;H} \vdash \mathtt{C}$ so this parent expression must also be well typed.

  * Case $\mathtt{L''.l}$. Then by T-Field $\mathtt{L''}$ must have a record type, contradicting the fact that $\mathtt{E;H} \vdash \mathtt{L''} : \mathtt{PrimitiveType}$.
  * Case $\mathtt{L''}$ is an immediate subexpression of an $\oplus$ operation. But these operations only apply to primitives, so the solver cannot satisfy them by assigning $\mathtt{x}$ to $\mathtt{r'}$.
  * Case $\rho\ \mathtt{L''}$. This constraint is only satisfied if $\mathtt{x}$ is a boolean, so the solver will not assign $\mathtt{x}$ to $\mathtt{r'}$.

- Case $\mathtt{v}$ is a reference $\mathtt{r}$. Then $\mathtt{r} = \mathtt{r'}$. Since $\mathtt{E;H} \vdash \mathtt{v} : \mathtt{T}$, by T-Ref we have that $\mathtt{H}(\mathtt{r}){=}\{\mathtt{l}_1{:}\mathtt{v}_1, \ldots, \mathtt{l}_n{:}\mathtt{v}_n\}$ and $\mathtt{E;H} \vdash \mathtt{v}_1 : \mathtt{T}_1 \cdots \mathtt{E;H} \vdash \mathtt{v}_n : \mathtt{T}_n$ and $\mathtt{T}$ is $\{\mathtt{l}_1{:}\mathtt{T}_1, \ldots, \mathtt{l}_n{:}\mathtt{T}_n\}$. By T-Field we have $\mathtt{E;H} \vdash \mathtt{L.l}_i : \mathtt{T}_i$ for each $1 \leq i \leq n$, so by induction $\mathtt{E';H'} \vdash \mathtt{L.l}_i : \mathtt{T}_i$ for each $1 \leq i \leq n$.

  Since stay($\mathtt{H}$, $\mathtt{required}$) $= \mathtt{C}_{H_s}$, by StayHeap we have that $\mathtt{C}_{H_s}$ includes a conjunct $\mathtt{C}_r$ such that stay($\mathtt{r}{=}\{\mathtt{l}_1{:}\mathtt{v}_1, \ldots, \mathtt{l}_n{:}\mathtt{v}_n\}$, $\mathtt{required}$) $= \mathtt{C}_r$. By StayObject $\mathtt{C}_r$ has the form $(\mathtt{required}\ \mathbb{H}(\mathtt{r}){=}\{\mathtt{l}_1{:}\mathtt{x}_1, \ldots, \mathtt{l}_n{:}\mathtt{x}_n\}) \wedge \mathtt{C}_1 \wedge \cdots \wedge \mathtt{C}_n$, where $\mathtt{x}_1 \ldots \mathtt{x}_n$ are fresh variables and stay($\mathtt{x}_i{=}\mathtt{v}_i$, $\mathtt{required}$) $= \mathtt{C}_i$. Therefore any solution to $(\mathtt{C'} \wedge \mathtt{C}_{E_s} \wedge \mathtt{C}_{H_s})$ must map $\mathtt{r}$ to an object value of the form $\{\mathtt{l}_1{:}\mathtt{v}'_1, \ldots, \mathtt{l}_n{:}\mathtt{v}'_n\}$ in $\mathtt{H'}$.

  Since $\mathtt{E';H'} \vdash \mathtt{L.l}_i : \mathtt{T}_i$ for each $1 \leq i \leq n$, by Lemma 3 we have $\mathtt{E;H} \vdash \mathtt{L.l}_i \Downarrow \mathtt{v}''_i$ and $\mathtt{E;H} \vdash \mathtt{v}''_i : \mathtt{T}_i$ for each $1 \leq i \leq n$. Since the last rule in each of these evaluation derivations must be E-Field we have that $\mathtt{v}''_i = \mathtt{v}'_i$ for each $1 \leq i \leq n$. Finally, since $\mathtt{E';H'} \vdash \mathtt{r} : \mathtt{T'}$, the last rule in this derivation must be T-Ref, so we have that $\mathtt{T'} = \{\mathtt{l}_1{:}\mathtt{T}_1, \ldots, \mathtt{l}_n{:}\mathtt{T}_n\}$.

□

Lemma 3 *If* $\mathtt{E;H} \vdash \mathtt{L} : \mathtt{T}$*, then there exists a value* $\mathtt{v}$ *such that* $\mathtt{E;H} \vdash \mathtt{L} \Downarrow \mathtt{v}$ *and* $\mathtt{E;H} \vdash \mathtt{v} : \mathtt{T}$*.*

Proof. By structural induction on L:

- Case L is a variable x: Then by T-VAR we have that E(x) = v and E;H ⊢ v : T. Finally by E-VAR we have E;H ⊢ x ⇓ v.
- Case L has the form L′.l$_i$: By T-FIELD we have that E;H ⊢ L′ : {l$_1$:T$_1$,...,l$_n$:T$_n$} and $1 \leq i \leq n$ and T = T$_i$. By induction there exists a value v′ such that E;H ⊢ L′ ⇓ v′ and E;H ⊢ v′ : {l$_1$:T$_1$,...,l$_n$:T$_n$}. Case analysis of the derivation of E;H ⊢ v′ : {l$_1$:T$_1$,...,l$_n$:T$_n$}:
  - Case T-CONST: Then {l$_1$:T$_1$,...,l$_n$:T$_n$} = PrimitiveType and we have a contradiction.
  - Case T-REF: Then we have v′ = r and H(r)={l$_1$:v$_1$,...,l$_n$:v$_n$} and E;H ⊢ v$_i$: T$_i$. Finally, by E-FIELD we have E;H ⊢ L′.l$_i$ ⇓ v$_i$.

□

Lemma 4 *If* E;H ⊢ C *and* solve(E, H, ℂ, required) = E′;H′ *and* E;H ⊢ L ⇓ v*, then there exists a value* v′ *such that* E′;H′ ⊢ L ⇓ v′*. Furthermore, if* v *is an object reference* r*, then also* v′ = r*.*

Proof. Since solve(E, H, ℂ, required) = E′;H′ by SOLVE we have that stay(E, required) = C$_{E_s}$ and stay(H, required) = C$_{H_s}$ and E;H ⊢ ℂ ⇝ C′ and E′;H′ ⊨ (C′ ∧ C$_{E_s}$ ∧ C$_{H_s}$). We proceed by structural induction on L:

- Case L is a variable x. Since E;H ⊢ L ⇓ v, by E-VAR we have that E(x) = v. Then since stay(E, required) = C$_{E_s}$, by STAYENV we have that C$_{E_s}$ includes a conjunct C$_x$ such that stay(x=v, required) = C$_x$. Case analysis of the rule used in the derivation of stay(x=v, required) = C$_x$:
  - STAYCONST: Then C$_x$ has the form weak x=v and v is a constant c. Therefore the variable x appears in the constraint (C′ ∧ C$_{E_s}$ ∧ C$_{H_s}$), so any solution to the constraint must include some value v′ for x in E′, and the result follows by E-VAR.
  - STAYREF: Then C$_x$ has the form required x=v and v is an object reference r. Therefore any solution to (C′ ∧ C$_{E_s}$ ∧ C$_{H_s}$) must map x to r in E′, and the result follows by E-VAR.
- Case L has the form L′.l$_i$. Then by E-FIELD we have E;H ⊢ L′ ⇓ r and H(r) = {l$_1$:v$_1$,...,l$_n$:v$_n$} and $1 \leq i \leq n$ and v = v$_i$. By induction we have E′;H′ ⊢ L′ ⇓ r. Since stay(H, required) = C$_{H_s}$, by STAYHEAP we have that C$_{H_s}$ includes a conjunct C$_r$ such that stay(r={l$_1$:v$_1$,...,l$_n$:v$_n$}, required) = C$_r$. By STAYOBJECT C$_r$ has the form (required ℍ(r)={l$_1$:x$_1$,...,l$_n$:x$_n$}) ∧ C$_1$ ∧ ⋯ ∧ C$_n$, where x$_1$ ... x$_n$ are fresh variables and stay(x$_i$=v$_i$, required) = C$_i$. Therefore any solution to (C′ ∧ C$_{E_s}$ ∧ C$_{H_s}$) must map r to an object value of the form {l$_1$:v′$_1$,...,l$_n$:v′$_n$} in H′, where v′$_i$ is the value assigned to variable x$_i$ by the solution. Therefore by E-FIELD we have E;H ⊢ L′.l$_i$ ⇓ v′$_i$. Finally suppose v$_i$ is an object reference r$_i$. Then by STAYREF C$_i$ has the form required x$_i$ = r$_i$ so any solution to the constraints must map x$_i$ to r$_i$, so also v′$_i$ is r$_i$.

□

Lemma 5 *If* E;H ⊢ L ⇓ v *then there is some type* T *such that* E;H ⊢ L : T *and* E;H ⊢ v : T*.*

Proof. By induction on the derivation of E;H ⊢ L ⇓ v:

- Case E-VARIABLE. Then L is a variable x and E(x) = v. Then by well formedness there is some type T such that E;H ⊢ x : T and by T-Var also E;H ⊢ v : T.
- Case E-FIELD. Then L has the form $L'.l_i$ and E;H ⊢ L′ ⇓ r and H(r) = $\{l_1 : v_1, \ldots, l_n : v_n\}$ and $1 \leq i \leq n$ and $v = v_i$. By induction there is some type T′ such that E;H ⊢ L′ : T′ and E;H ⊢ r : T′. Then by T-REF T′ has the form $\{l_1 : T_1, \ldots, l_n : T_n\}$ where E;H ⊢ v_i : T_i. Then by T-FIELD also E;H ⊢ L : T_i.

□

Lemma 6 *If* E;H ⊢ L ⇓ r *then* r *is in the domain of* H.

Proof. Case analysis on the last rule in the evaluation derivation.

- Case E-VARIABLE. Then L is a variable x and E(x) = r. Then the result follows from the assumption that E and H are well formed.
- Case E-FIELD. Then L has the form $L'.l_i$ and E;H ⊢ L′ ⇓ r′ and H(r) = $\{l_1 : v_1, \ldots, l_n : v_n\}$ and $1 \leq i \leq n$ and r = v_i. Then the result follows from the assumption that E and H are well formed.

□

The second theorem formalizes the idea that all solutions to an assignment will produce structurally equivalent environments and heaps:

Theorem 2 *(Structural Determinism) If* <E|H|ℂ|𝕀|L := e> ⟶ <E_1|H_1|$\mathbb{C}_1$|$\mathbb{I}_1$> *and* <E|H|ℂ|𝕀|L := e> ⟶ <E_2|H_2|$\mathbb{C}_2$|$\mathbb{I}_2$> *and* E;H ⊢ x : T_0*, then there exists a type* T *such that* E_1;H_1 ⊢ x : T *and* E_2;H_2 ⊢ x : T.

Proof. By S-ASGN we have E;H ⊢ e ⇓ v and stayPrefix(E, H, L) = C_L and stayPrefix(E, H, I) = C_I and solve(E, H, $C_L \wedge C_I \wedge$ L=v ∧ I, weak) = E_1';H_1' and E_1';H_1' ⊢ C and solve(E_1', H_1', ℂ ∧ L=v, required) = E_1;H_1. Also by S-ASGN and Lemma 13 we have solve(E, H, $C_L \wedge C_I \wedge$ L=v ∧ I, weak) = E_2';H_2' and E_2';H_2' ⊢ C and solve(E_2', H_2', ℂ ∧ L=v, required) = E_2;H_2. By Lemma 7 we have that $E_1' = E_2'$ and $H_1' = H_2'$. Since E;H ⊢ x : T_0, by T-VAR x is in the domain of E, so it is also in the domain of E_1' and by E-VAR we have E_1';H_1' ⊢ x ⇓ v_x where E_1'(x) = v_x. Then by Lemma 5 there is some type T′ such that E_1';H_1' ⊢ v_x : T′ and E_1';H_1' ⊢ x : T′. Finally by Lemma 2 we have that E_1;H_1 ⊢ x : T′ and E_2;H_2 ⊢ x : T′, so the result is shown with T = T′.

□

Definition 1 *We say that* L_1 *and* L_2 *are* aliases *given* E *and* H *if either:*

- L_1 *and* L_2 *are the same variable* x
- L_1 *has the form* $L_1'.l$ *and* L_2 *has the form* $L_2'.l$ *and there is a reference* r *such that* E;H ⊢ L_1' ⇓ r *and* E;H ⊢ L_2' ⇓ r

Definition 2 *We say that* L *and* L′ *are the* operands *of the constraint* L == L′.

Definition 3 *We define the* induced graph *of* I *and* L *given* E *and* H *as follows. Let S be the set that includes* L *as well as all operands of identity tests in* I*. Partition S into equivalence classes defined by the alias relation:* $\mathtt{L}_1$ *and* $\mathtt{L}_2$ *are in the same equivalence class if and only if they are aliases given* E *and* H*. Then the induced graph has one node per equivalence class and an undirected edge between nodes N_1 and N_2 if there is a conjunct* $\mathtt{L}_1\mathtt{==}\mathtt{L}_2$ *in* I *such that* $\mathtt{L}_1$ *belongs to node N_1 and* $\mathtt{L}_2$ *belongs to node N_2.*

Definition 4 *We say that a node in the induced graph of* I *and* L *given* E *and* H *is* relevant *if it is reachable from* L*'s node in the graph; an l-value is* relevant *if its node in the graph is relevant.*

Definition 5 *The* relevant update *of* E *and* H *for* I *and* L=v *is the environment* $\mathtt{E}'$ *and heap* $\mathtt{H}'$ *that are identical to* E *and* H *except that for each relevant l-value* $\mathtt{L}_0$ *in the induced graph of* I *and* L *given* E *and* H*:*

- *If* $\mathtt{L}_0$ *is a variable* x*, then* $\mathtt{E}'(\mathtt{x}) = \mathtt{v}$*.*
- *If* $\mathtt{L}_0$ *has the form* $\mathtt{L}_0'.\mathtt{l}$ *and* $\mathtt{E};\mathtt{H} \vdash \mathtt{L}_0' \Downarrow \mathtt{r}$*, then* $\mathtt{E}';\mathtt{H}' \vdash \mathtt{r.l} \Downarrow \mathtt{v}$*.*

Lemma 7 *If* stayPrefix(E, H, L) = $\mathtt{C}_L$ *and* stayPrefix(E, H, I) = $\mathtt{C}_I$ *and* solve(E, H, $\mathtt{C}_L \wedge \mathtt{C}_I \wedge \mathtt{L{=}v} \wedge \mathtt{I}$, weak) = $\mathtt{E}_1;\mathtt{H}_1$ *and* solve(E, H, $\mathtt{C}_L \wedge \mathtt{C}_I \wedge \mathtt{L{=}v} \wedge \mathtt{I}$, weak) = $\mathtt{E}_2;\mathtt{H}_2$*, then* $\mathtt{E}_1 = \mathtt{E}_2$ *and* $\mathtt{H}_1 = \mathtt{H}_2$*.*

Proof. By Solve we have stay(E, weak) = $\mathtt{C}_E$ and stay(H, weak) = $\mathtt{C}_H$ and $\mathtt{E};\mathtt{H} \vdash (\mathtt{C}_L \wedge \mathtt{C}_I \wedge \mathtt{L{=}v} \wedge \mathtt{I}) \rightsquigarrow \mathtt{C}'$ and $\mathtt{E}';\mathtt{H}' \models (\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H)$ and $\mathtt{E}'';\mathtt{H}'' \models (\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H)$.

By Lemma 9, $\mathtt{E}'$ and $\mathtt{H}'$ include all the updates of the relevant update (which we will call $\mathtt{E}_0$ and $\mathtt{H}_0$) of E and H for I and L=v, and similarly for $\mathtt{E}''$ and $\mathtt{H}''$. To complete the proof we argue that both of these solutions are in fact identical to the relevant update. WLOG we consider $\mathtt{E}'$ and $\mathtt{H}'$. By Lemma 8 the relevant update of E and H for I and L=v is a solution to the constraint $\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H$. Therefore if $\mathtt{E}'$ and $\mathtt{H}'$ is not the relevant update, then by Definition 5 either:

- there is a variable x such that $\mathtt{E}_0(\mathtt{x}) = \mathtt{E}(\mathtt{x})$ but $\mathtt{E}'(\mathtt{x})$ has a different value
- there is a reference r and field label l such that $\mathtt{H}_0(\mathtt{r}).\mathtt{l} = \mathtt{H}(\mathtt{r}).\mathtt{l}$ but $\mathtt{H}'(\mathtt{r}).\mathtt{l}$ has a different value

Consider the former. Since stay(E, weak) = $\mathtt{C}_E$, by StayEnv, StayConst, and StayRef we have that $\mathtt{C}_E$ includes a weak constraint $\mathtt{x{=}v}_x$, where $\mathtt{E}(\mathtt{x}) = \mathtt{v}_x$. Since $\mathtt{E}'$ and $\mathtt{H}'$ include all the updates of the relevant update, $\mathtt{E}'$ and $\mathtt{H}'$ satisfy strictly fewer weak constraints than the relevant update, contradicting the optimality of $\mathtt{E}'$ and $\mathtt{H}'$.

Similarly, consider the latter. Since stay(H, weak) = $\mathtt{C}_H$, by StayHeap and StayObject $\mathtt{C}_H$ includes a required constraint $\mathbb{H}(\mathtt{r}){=}\{\mathtt{l}_1{:}\mathtt{x}_1, \ldots, \mathtt{l}_n{:}\mathtt{x}_n\}$ where the $\mathtt{x}_i$ variables are fresh and l is some $\mathtt{l}_i$. Then by StayConst and StayRef there is a weak constraint of the form $\mathtt{x}_i$ = $\mathtt{v}_i$ where $\mathtt{H}(\mathtt{r}).\mathtt{l}_i = \mathtt{v}_i$. Since $\mathtt{E}'$ and $\mathtt{H}'$ include all the updates of the relevant update, $\mathtt{E}'$ and $\mathtt{H}'$ satisfy strictly fewer weak constraints than the relevant update, contradicting the optimality of $\mathtt{E}'$ and $\mathtt{H}'$.

□

Lemma 8 *If* stay(E, weak) = $\mathtt{C}_E$ *and* stay(H, weak) = $\mathtt{C}_H$ *and* stayPrefix(E, H, L) = $\mathtt{C}_L$ *and* stayPrefix(E, H, I) = $\mathtt{C}_I$ *and* $\mathtt{E};\mathtt{H} \vdash (\mathtt{C}_L \wedge \mathtt{C}_I \wedge \mathtt{L{=}v} \wedge \mathtt{I}) \rightsquigarrow \mathtt{C}'$ *and the constraint* $\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H$ *is satisfiable, then the relevant update* $\mathtt{E}'$ *and* $\mathtt{H}'$ *of* E *and* H *for* I *and* L=v *is a solution to the constraint* $\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H$*.*

Proof. It suffices to show that all required constraints in $\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H$ are satisfied in $\mathtt{E}'$ and $\mathtt{H}'$. We consider the various constraints in turn:

- $\mathtt{C}_E$: Since stay(E, weak) = $\mathtt{C}_E$, by StayEnv, StayConst, and StayRef there are no required constraints in $\mathtt{C}_E$, so all required constraints are satisfied vacuously.
- $\mathtt{C}_H$: Since stay(H, weak) = $\mathtt{C}_H$, by StayHeap, StayObject, StayConst, and StayRef the only required constraints in $\mathtt{C}_H$ have the form $\mathbb{H}(\mathtt{r})=\{\mathtt{l}_1:\mathtt{x}_1,\ldots,\mathtt{l}_n:\mathtt{x}_n\}$ where the $\mathtt{x}_i$ variables are fresh and H maps r to some value of the form $\{\mathtt{l}_1:\mathtt{v}_1,\ldots,\mathtt{l}_n:\mathtt{v}_n\}$. By Definition 5 also $\mathtt{H}'$ maps r to a value of the form $\{\mathtt{l}_1:\mathtt{v}'_1,\ldots,\mathtt{l}_n:\mathtt{v}'_n\}$ so the constraint is satisfied.
- $\mathtt{C}_L$: By StayPrefixField the conjuncts in C_L have the form x=v or r.l=v. Suppose the relevant update fails to satisfy one of these conjuncts. We consider each form in turn:
  - x=v: Then the relevant update maps x to some $\mathtt{v}' \neq \mathtt{v}$ in $\mathtt{E}'$. But by Solve and Lemma 9 any solution to the constraint $\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H$ must map x to $\mathtt{v}'$ in the environment, which violates the constraint x=v. So there must be no solution to the constraints, contradicting our assumption of satisfiability.
  - r.l=v: Then the relevant update maps r.l to some $\mathtt{v}' \neq \mathtt{v}$ in $\mathtt{H}'$. But by Solve and Lemma 9 any solution to the constraint $\mathtt{C}' \wedge \mathtt{C}_E \wedge \mathtt{C}_H$ must map r.l to $\mathtt{v}'$ in the environment, which violates the constraint r.l=v. So there must be no solution to the constraints, contradicting our assumption of satisfiability.
- $\mathtt{C}_I$: By StayPrefixIdent the conjuncts in C_L have the form x=v or r.l=v. So the argument is identical to that above for the $\mathtt{C}_L$ constraint.
- L=v: By Definitions 3 and 5 we have that L is relevant for I and L given E and H. Then by Lemma 10 we have $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L} \Downarrow \mathtt{v}$, so by E-Op and the semantics of equality we have $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L=v} \Downarrow \mathtt{true}$.
- I: Consider a conjunct $\mathtt{L}_0$ == $\mathtt{L}_1$ in I. We have two cases:
  - $\mathtt{L}_0$ is relevant for I and L given E and H. By Definitions 4 and 3, $\mathtt{L}_1$ is also relevant. Then by Lemma 10 we have $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L}_0 \Downarrow \mathtt{v}$ and $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L}_1 \Downarrow \mathtt{v}$, so by E-IdentityTrue we have $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L}_0\mathtt{==}\mathtt{L}_1 \Downarrow \mathtt{true}$.
  - $\mathtt{L}_0$ is not relevant for I and L given E and H. By Definitions 4 and 3, $\mathtt{L}_1$ is also not relevant. By well formedness we have $\mathtt{E};\mathtt{H} \vdash \mathtt{L}_0\mathtt{==}\mathtt{L}_1 \Downarrow \mathtt{true}$, so by E-IdentityTrue there is some $\mathtt{v}_0$ such that $\mathtt{E};\mathtt{H} \vdash \mathtt{L}_0 \Downarrow \mathtt{v}_0$ and $\mathtt{E};\mathtt{H} \vdash \mathtt{L}_1 \Downarrow \mathtt{v}_0$. Then by Lemma 11 we have $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L}_0 \Downarrow \mathtt{v}_0$ and $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L}_1 \Downarrow \mathtt{v}_0$, so by E-IdentityTrue we have $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L}_0\mathtt{==}\mathtt{L}_1 \Downarrow \mathtt{true}$.

□

Lemma 9 *If* stayPrefix(E, H, L) = $\mathtt{C}_L$ *and* stayPrefix(E, H, I) = $\mathtt{C}_I$ *and* solve(E, H, $\mathtt{C}_L \wedge \mathtt{C}_I \wedge \mathtt{L=v} \wedge \mathtt{I}$, weak) = $\mathtt{E}';\mathtt{H}'$ *and* $\mathtt{L}_0$ *is a relevant l-value of* I *and* L *given* E *and* H*, then* $\mathtt{E}';\mathtt{H}' \vdash \mathtt{L}_0 \Downarrow \mathtt{v}$ *and:*

- *If* $\mathtt{L}_0$ *is a variable* x*, then* $\mathtt{E}'(\mathtt{x}) = \mathtt{v}$.
- *If* $\mathtt{L}_0$ *has the form* $\mathtt{L}'_0.\mathtt{l}$ *and* $\mathtt{E};\mathtt{H} \vdash \mathtt{L}'_0 \Downarrow \mathtt{r}$*, then* $\mathtt{E}';\mathtt{H}' \vdash \mathtt{r.l} \Downarrow \mathtt{v}$.

Proof.

By Definition 4, we know that $\mathtt{L}_0$'s node in the induced graph for I and L given E and H is reachable from L's node. The proof proceeds by induction on the length k of the path between these nodes.

- Case $k = 0$. Then by Definition 3, L and L_0 are aliases given E and H. Case analysis on the structure of L:
  - Case L is a variable x. Then by Definition 1 also L_0 is x. Since solve(E, H, $C_L \wedge C_I \wedge$ L=v $\wedge$ I, weak) = E′;H′ we must have E′;H′ ⊢ x=v ⇓ true, so by E-Op and the semantics of equality we have E′;H′ ⊢ x ⇓ v. Then by E-Var also E′(x) = v.
  - Case L has the form L′.l. Then by Definition 1 also L_0 has the form L_0'.l and E;H ⊢ L′ ⇓ r and E;H ⊢ L_0' ⇓ r. We are given stayPrefix(E, H, L) = C_L. Then since solve(E, H, $C_L \wedge C_I \wedge$ L=v $\wedge$ I, weak) = E′;H′ we must have E′;H′ ⊢ C_L ⇓ true. Similarly we are given stayPrefix(E, H, I) = C_I so by StayPrefixIdent we have also stayPrefix(E, H, L_0) = C_{L_0} where C_{L_0} is a conjunct within C_I. Then since solve(E, H, $C_L \wedge C_I \wedge$ L=v $\wedge$ I, weak) = E′;H′ we must have E′;H′ ⊢ C_{L_0} ⇓ true. So by Lemma 13 and Lemma 12 we have E′;H′ ⊢ L′ ⇓ r and E′;H′ ⊢ L_0' ⇓ r. We also know E′;H′ ⊢ L=v ⇓ true, so by E-Op and the semantics of equality we have E′;H′ ⊢ L ⇓ v. Then by E-Field we have H′(r) is a record whose l field has value v. Then by E-Field also E′;H′ ⊢ r.l ⇓ v and E′;H′ ⊢ L_0 ⇓ v.
- Case $k > 0$. Then by Definition 3 there is some neighbor node of L_0's node that is only $k - 1$ hops away from L's node, which contains an l-value L_1' such that L_0' == L_1' or vice versa is in I, where L_0' is an alias of L_0 given E and H. By induction we have E′;H′ ⊢ L_1' ⇓ v. Then since the identities in I are satisfied in E′ and H′, by E-IdentityTrue also E′;H′ ⊢ L_0' ⇓ v. Case analysis on the structure of L_0':
  - Case L_0' is a variable x. Then by Definition 1 also L_0 is x and by E-Var also E′(x) = v.
  - Case L_0' has the form L′.l. Then by Definition 1 also L_0 has the form L_{00}.l and E;H ⊢ L′ ⇓ r and E;H ⊢ L_{00} ⇓ r. We are given stayPrefix(E, H, I) = C_I so by StayPrefixIdent we have also stayPrefix(E, H, L_0') = $C_{L_0'}$ where $C_{L_0'}$ is a conjunct within C_I. Then since solve(E, H, $C_L \wedge C_I \wedge$ L=v $\wedge$ I, weak) = E′;H′ we must have E′;H′ ⊢ $C_{L_0'}$ ⇓ true. By the same argument we must also have E′;H′ ⊢ C_{L_0} ⇓ true where stayPrefix(E, H, L_0) = C_{L_0}. So by Lemma 13 and Lemma 12 we have E′;H′ ⊢ L′ ⇓ r and E′;H′ ⊢ L_{00} ⇓ r. Then since E′;H′ ⊢ L_0' ⇓ v by E-Field we have H′(r) is a record whose l field has value v. Then by E-Field also E′;H′ ⊢ r.l ⇓ v and E′;H′ ⊢ L_0 ⇓ v.

□

Lemma 10 *If* stayPrefix(E, H, L) = C_L *and* stayPrefix(E, H, I) = C_I *and* E′ *and* H′ *is the relevant update of* E *and* H *for* I *and* L=v *and* E′;H′ ⊢ C_L ⇓ true *and and* E′;H′ ⊢ C_I ⇓ true *and* L_0 *is a relevant l-value for* I *and* L *given* E *and* H*, then* E′;H′ ⊢ L_0 ⇓ v*.*

Proof. Case analysis of the structure of L_0:

- Case L_0 is a variable x: By Definition 5 E′(x) = v, so the result follows by E-Var.
- Case L_0 has the form L′.l: First we argue that stayPrefix(E, H, L_0) = C_{L0} and E′;H′ ⊢ C_{L0} ⇓ true. If L_0 is L, then these follow from the assumptions of the lemma. Otherwise, by Definition 4 L_0 is an operand in an identity test in I. Then since stayPrefix(E, H, I) = C_I by StayPrefixIdent we have stayPrefix(E, H, L_0) = C_{L0}, where C_{L0} is a conjunct in C_I. Then since E′;H′ ⊢ C_I ⇓ true also E′;H′ ⊢ C_{L0} ⇓ true.

  Then by Lemma 12 there is some r′ such that E;H ⊢ L′ ⇓ r′ and E′;H′ ⊢ L′ ⇓ r′. By Lemma 13 and Definition 5 we have E;H ⊢ r′.l ⇓ v and the result follows by E-Field.

□

Lemma 11 *If* stayPrefix(E, H, L) = C_L *and* stayPrefix(E, H, I) = C_I *and* E′ *and* H′ *is the relevant update of* E *and* H *for* I *and* L=v *and* E′;H′ ⊢ C_L ⇓ true *and and* E′;H′ ⊢ C_I ⇓ true *and* L_0 *is an operand of an identity test in* I *and* L_0 *is not relevant for* I *and* L *given* E *and* H *and* E;H ⊢ L_0 ⇓ v_0, E′;H′ ⊢ L_0 ⇓ v_0.

Proof. Case analysis on the structure of L_0:

- Case L_0 is a variable x. Since x is not relevant, by Definition 5 the value of x in E′ is the same as that in Euid. Since E;H ⊢ L_0 ⇓ v_0, by E-VAR we have E(x) = v_0, so also E′(x) = v_0 and the result follows by E-VAR.
- Case L_0 has the form L′.l. Since L_0 is an operand in an identity test in I and stayPrefix(E, H, I) = C_I, by STAYPREFIXIDENT we have stayPrefix(E, H, L_0) = C_{L0}, where C_{L0} is a conjunct in C_I. Then since E′;H′ ⊢ C_I ⇓ true also E′;H′ ⊢ C_{L0} ⇓ true.

  Therefore by Lemma 12 there is some r′ such that E;H ⊢ L′ ⇓ r′ and E′;H′ ⊢ L′ ⇓ r′. Since E;H ⊢ L_0 ⇓ v_0 by E-FIELD and Lemma 13 we have that $\mathbb{H}$(r′).l is v_0. By Definition 5 H′(r′) also has a field with label l, so by E-FIELD we are done if that field's value is v_0. Suppose not. Then by Definition 5 there is some relevant l-value L_1 of the form L_1'.l such that E;H ⊢ L_1' ⇓ r′. But then by Definition 1 we have that L_0 and L_1 are aliases so they correspond to the same node in the induced graph of I and L given E and H by Definition 3. But then since L_1 is relevant, by Definition 4 so is L_0 and we have a contradiction.

□

Lemma 12 *If* stayPrefix(E, H, L.f) = C *and* E′;H′ ⊢ C ⇓ true*, then there is some reference* r *such that* E;H ⊢ L ⇓ r *and* E′;H′ ⊢ L ⇓ r.

Proof. By STAYPREFIXFIELD we have L.f = x.l_1.....l_n and $n > 0$ and E;H ⊢ x ⇓ r and E;H ⊢ x.l_1 ⇓ r_1 ⋯ E;H ⊢ x.l_1.....l_{n-1} ⇓ r_{n-1} and C is x=r ∧ x.l_1=r_1 ∧ ⋯ ∧ x.l_1.....l_{n-1}=r_{n-1}.

- Case $n = 1$. Then L is x and C is x=r. Since E′;H′ ⊢ C ⇓ true, by E-OP and the semantics of equality we have E′;H′ ⊢ x ⇓ v_1 and E′;H′ ⊢ r ⇓ v_2 and $v_1 = v_2$. By E-REF we have that v_2 = r, so the result follows.
- Case $n > 1$. Then L is x.l_1.....l_{n-1}. Since E′;H′ ⊢ C ⇓ true, by E-OP and the semantics of equality we have E′;H′ ⊢ L ⇓ v_1 and E′;H′ ⊢ r_{n-1} ⇓ v_2 and $v_1 = v_2$. By E-REF we have that $v_2 = r_{n-1}$, so the result follows.

□

Lemma 13 *(Determinism)*

- *If* E;H ⊢ *e* ⇓ v_1 *and* E;H ⊢ *e* ⇓ v_2 *then* $v_1 = v_2$.
- *If stay(*E, *ρ) =* C_1 *and stay(*E, *ρ) =* C_2 *then* $C_1 = C_2$.
- *If stay(*H, *ρ) =* C_1 *and stay(*H, *ρ) =* C_2 *then* $C_1 = C_2$.
- *If stayPrefix(*E, H, *L) =* C_1 *and stayPrefix(*E, H, *L) =* C_2 *then* $C_1 = C_2$.

- *If stayPrefix(*$\mathtt{E}$, $\mathtt{H}$, $\mathcal{I}$*)* $= \mathcal{C}_1$ *and stayPrefix(*$\mathtt{E}$, $\mathtt{H}$, $\mathcal{I}$*)* $= \mathcal{C}_2$ *then* $\mathtt{C}_1 = \mathtt{C}_2$.

Proof. Straightforward. □