Quirrel Specification

Daniel Spiewak

February 29, 2012

Abstract

This is the formal specification for the Quirrel language. Specifically, this document contains the syntactic and lexical specification, the formal type and type constraint rules. This document also contains an informal specification of the evaluation of each of the syntactic constructs, as well as a working set of built-in functions. Finally, we include a formal meta-theoretic framework describing the semantics and meaning of the major constructs which define the language.

Contents

1 Grammar

Typographical conventions:

- Non-terminals are rendered in a normal typeface
- Terminals are rendered using `typewriter font`
- Lexical primitives are rendered using CAPITALS
- Regular expressions are delimited by /

This grammar is intended to be unambiguous in all respects with the exception of precedence and associativity. This is done because these two properties are messy to encode in a grammar and thus are better left to an auxiliary specification.

```
expr  ::=  IDENTIFIER ( formals ) := expr expr
        |  IDENTIFIER := expr expr

        |  new expr
        |  relations expr
```

```
          |   ns-id
          |   TIC-IDENTIFIER

          |   PATH-LITERAL
          |   STRING-LITERAL
          |   NUM-LITERAL
          |   BOOL-LITERAL

          |   { properties }
          |   [ nullable-actuals ]
          |   expr . PROPERTY
          |   expr [ expr ]

          |   ns-id ( actuals )
          |   expr where expr
          |   expr with expr

          |   expr + expr
          |   expr - expr
          |   expr * expr
          |   expr / expr

          |   expr < expr
          |   expr <= expr
          |   expr > expr
          |   expr <= expr

          |   expr = expr
          |   expr != expr

          |   expr & expr
          |   expr | expr

          |   ! expr
          |   neg expr

          |   ( expr )

  ns-id  ::=  ns-id :: IDENTIFIER
          |   IDENTIFIER

formals  ::=  formals , TIC-IDENTIFIER
          |   TIC-IDENTIFIER
```

```
       relations  ::=  relations ~ expr
                   |   expr ~ expr

         actuals  ::=  actuals , expr
                   |   expr

nullable-actuals  ::=  actuals
                   |   ϵ

      properties  ::=  properties , property
                   |   property
                   |   ϵ

        property  ::=  PROPERTY : expr

      IDENTIFIER  ::=  /[a-zA-Z_]['a-zA-Z_0-9]*/ \ { false, new, true, where, with, neg }

  TIC-IDENTIFIER  ::=  /'[a-zA-Z_0-9]['a-zA-Z_0-9]*/

        PROPERTY  ::=  /[a-zA-Z_][a-zA-Z_0-9]*/

    PATH-LITERAL  ::=  /(//[a-zA-Z_\-0-9]+)+ /

  STRING-LITERAL  ::=  /"([^"\n\r\\]|\\.)*"/

     NUM-LITERAL  ::=  /[0-9]+(.[0-9]+)?([eE][0-9]+)?/

    BOOL-LITERAL  ::=  true
                   |   false

      WHITESPACE  ::=  /([;\s]+|--.*|(-([^\-]|-+[^)\-])*-))+/
```

1.1 Precedence and Associativity

Operator precedence flows *down* the table. Thus, operators that are higher in the table have precedence over operators that are lower in the table.

Operator(s)	**Precedence**
`.`, `[]`	N/A
`!`, `neg`	N/A
`*`, `/`	LEFT
`+`, `-`	LEFT
`<`, `<=`, `>`, `>=`	LEFT

`=`, `!=`	Left
`&`, `|`	Left
`new`	N/A
`where`	Left
~	None

Additionally, array declaration literals and the array dereference operator do not associate. Thus, the text "`[a][0]`" will be rejected as invalid by the parser.

2 Informal Semantics

All expressions in Quirrel produce sets of events. This holds for even literal value expressions (*e.g.* `42`), which will be implicitly lifted into singleton sets.

2.1 Characteristic Functions

```
expr    ::= IDENTIFIER ( formals ) := expr expr
          | IDENTIFIER := expr expr

formals ::= formals , TIC-IDENTIFIER
          | TIC-IDENTIFIER
```

Defines a characteristic function of zero or more parameters according to the values of the first expression within the lexical scope defined by the second expression. Characteristic functions are sets of events. Parameters of characteristic functions are universally-quantified when left unspecified. Thus, a characteristic function defined as `foo('userId) := ...` may be used as merely `foo` where the result will be the set of values produced by the defining expression (to the right of the `:=`) when evaluated for *all* possible values for `'userId`. This set may or may not be finite, depending on the constraints placed upon `'userId` by its usage within the defining expression. If the set is infinite, it may not be returned as the final result of a query. It may, however, be used as an intermediate result.

Characteristic functions *always* represent sets, regardless of whether or not their parameters are fully specified. As all expressions produce sets, the defining expression for a characteristic function must always produce a set. Sets of sets are not legal in Quirrel. Thus, a characteristic function where the defining expression produces a set will result in a set containing all of the values from the produced sets. In effect, the meaningless set of sets is flattened into a single set.

It is not possible to define a characteristic function on a set. Tic variables must always represent a *single* event. When used in expressions, any tic variable will be implicitly lifted into a singleton set containing its event. The reason for this restriction is to ensure that the predicate calculus of Quirrel remains first-order.

2.2 New

```
expr ::= new expr
```

Produces a set with the same values as the given set but with entirely new identities. This has the effect of introducing a set with the same events as the given set but unrelated to any of the sets in the given set's relational family.

2.3 Relation

$$\begin{array}{rcl} \text{expr} & ::= & \text{relations expr} \\ \\ \text{relations} & ::= & \text{relations} \sim \text{expr} \\ & | & \text{expr} \sim \text{expr} \end{array}$$

Introduces a new relation between the set represented by the the first expression and the set represented by the second expression within the scope of the third expression. This relation is defined by the cross-product of the two sets. Thus, this will relate every event in the first set to every event in the second set.

If more than two sets are defined to be the constraining sets for a relation (*e.g.* `set1 :: set2 :: set3 expr`), it is considered equivalent to relating the first pair of sets, then relating the second set with the third in the scope of the first relation, and so on. Thus, the given example is purely syntactic sugar for the following: `set1 :: set2 set2 :: set3 expr`. Any number (greater than or equal to 2) of sets may be related in this fashion.

Note that relations are commutative and transitive. This means that relating two sets may have extensive consequences within the scope where that relation is defined. For example, suppose we have sets A, A', B and B', where A is related to A' and B is related to B'. If we introduce a new relation between A' and B', we have transitively also related A to B', A' to B and even A to B. This is why it is so important that relations are lexically scoped.

Because of the chaining that occurs in relation definition, it is very easy to accidentally introduce ambiguities if defining a relation for sets that are already related. For this reason, the relation operator is *not* defined for sets that are already related in the enclosing scope.

2.4 Variables

$$\begin{array}{rcl} \text{expr} & ::= & \text{ns-id} \\ & | & \text{TIC-IDENTIFIER} \\ \\ \text{ns-id} & ::= & \text{ns-id} :: \text{IDENTIFIER} \\ & | & \text{IDENTIFIER} \end{array}$$

Dereferences the specified variable within the current lexical scope and produces the relevant set. Note that identifiers always refer to sets while tic variables always refer to events. As all operations are applied to sets and not individual events, the results of dereferencing a tic variable will be implicitly lifted into a singleton set consisting of just that event. That singleton set will have a relation defined for any set with which it is used. Note that variable dereferencing is actually a zero-argument application of a characteristic function (see Section 2.8).

2.5 Values

As operations are applied to sets and not individual values, all values will be implicitly lifted, first into an event with bottom identity, and from there into a singleton set containing just that event.

2.5.1 Strings

$$\begin{array}{rcl} \text{expr} & ::= & \textsc{Path-Literal} \\ & | & \textsc{String-Literal} \end{array}$$

Path literals and string literals both produce JSON string values, where the value is the contents of the string literal or the full lexical value of the path literal. Thus, `//clicks` and `"/clicks"` are equivalent values.

2.5.2 Numbers

$$\begin{array}{rcl} \text{expr} & ::= & \textsc{Num-Literal} \end{array}$$

Produces a JSON numeric value.

2.5.3 Booleans

$$\begin{array}{rcl} \text{expr} & ::= & \textsc{Bool-Literal} \end{array}$$

Produces a JSON boolean value.

2.6 Objects

$$\begin{array}{rcl} \text{expr} & ::= & \texttt{\{}\ \text{properties}\ \texttt{\}} \\ \\ \text{properties} & ::= & \text{properties}\ \texttt{,}\ \text{property} \\ & | & \text{property} \\ & | & \epsilon \\ \\ \text{property} & ::= & \textsc{Property}\ \texttt{:}\ \text{expr} \end{array}$$

Produces a set of JSON objects with the specified properties. It is interesting to note that object construction does not produce a value, but rather a set of events. This is because object construction is an operation, not unlike any other infix operation like `+`, `=` or `where`. This is slightly unusual as object construction is an nary mixfix operation where the operands are the values produced by the property expressions.

2.7 Arrays

$$\begin{array}{rcl}
\text{expr} & ::= & \texttt{[}\ \text{nullable-actuals}\ \texttt{]} \\
\\
\text{actuals} & ::= & \text{actuals}\ \texttt{,}\ \text{expr} \\
 & | & \text{expr} \\
\\
\text{nullable-actuals} & ::= & \text{actuals} \\
 & | & \epsilon
\end{array}$$

Produces a set of JSON arrays with the specified values. It is interesting to note that array construction does not produce a value, but rather a set of events. This is because array construction is an operation, not unlike any other infix operation like `+`, `=` or `where`. This is slightly unusual as array construction is an nary mixfix operation where the operands are the values produced by the element expressions.

2.8 Functions

$$\begin{array}{rcl}
\text{expr} & ::= & \text{ns-id}\ \texttt{(}\ \text{actuals}\ \texttt{)} \\
\\
\text{ns-id} & ::= & \text{ns-id}\ \texttt{::}\ \textsc{Identifier} \\
 & | & \textsc{Identifier} \\
\\
\text{actuals} & ::= & \text{actuals}\ \texttt{,}\ \text{expr} \\
 & | & \text{expr}
\end{array}$$

Applies a function to the specified arguments, producing the resulting set. Any parameters which are left unspecified will be universally quantified with respect to the constraints imposed by the definition of the characteristic function. Note that function application with zero parameters has a special syntax that obviates the parentheses (see Section 2.4).

Note that this "argument omission" semantic is only valid for characteristic functions defined by the user. Built-in functions provided by the language must be fully applied, meaning all their parameters must be specified. For a list of built-in functions, see Section 3.

2.9 Operations

$$\text{expr} \quad ::= \quad \text{expr}\ \textsc{Identifier}\ \text{expr}$$

Maps the specified built-in operation over the two sets in a pair-wise fashion, producing a single set as a result. If there is no relation in scope which defines the pair-wise mapping for these two sets, the result will be the empty set. Relations are introduced into scope by other operations, binary and unary, as well as inferred from the rules of relational behavior (Section 6.4). Relations may also be introduced by explicit use of the relate operator (Section 2.3). A binary operation introduces two new relations onto the resulting set: one for each operand.

All binary and unary operations in QUIRREL behave in a similar fashion: mapping over their operands according to the most specific relation in scope for the operand sets. It is not possible for users to define custom operations; the only operations available in QUIRREL are the set of built-in operations listed in Section 4.

2.9.1 Arithmetic

$$\begin{array}{rcl} \text{expr} & ::= & \text{expr } \texttt{+} \text{ expr} \\ & | & \text{expr } \texttt{-} \text{ expr} \\ & | & \text{expr } \texttt{*} \text{ expr} \\ & | & \text{expr } \texttt{/} \text{ expr} \end{array}$$

Maps the specified arithmetic operation over the two sets after the fashion detailed in Section 2.9. If the result is undefined for a particular pair of values in the sets (e.g. `s1 / s2` where `s2` contains `0`), that pair will lack a corresponding value in the result set.

2.9.2 Boolean

$$\begin{array}{rcl} \text{expr} & ::= & \text{expr } \texttt{<} \text{ expr} \\ & | & \text{expr } \texttt{<=} \text{ expr} \\ & | & \text{expr } \texttt{>} \text{ expr} \\ & | & \text{expr } \texttt{<=} \text{ expr} \\ & & \\ & | & \text{expr } \texttt{=} \text{ expr} \\ & | & \text{expr } \texttt{!=} \text{ expr} \\ & & \\ & | & \text{expr } \texttt{\&} \text{ expr} \\ & | & \text{expr } \texttt{|} \text{ expr} \end{array}$$

Maps the specified boolean operation over the two sets after the fashion detailed in Section 2.9. Thus, operations such as `=` are not in fact set equality but pair-wise value equality *within* the sets (though they behave very much like set equality in the absence of an explicit relation between the operand sets).

The two equality operations (`=` and `!=`) are defined on all JSON values in a structural fashion. Thus, two events which consist of the same data will be considered equal even though their identities are distinct.

2.10 Unary Operations

$$\begin{array}{rcl} \text{expr} & ::= & \texttt{!} \text{ expr} \\ & | & \sim \text{expr} \end{array}$$

Maps the specified operation over the set similar to the way in which binary operations are mapped (Section 2.9). Unary operations are slightly simpler in that they do not require a relation to be in scope since they are only mapping over a *single* set. However, like binary operations, unary operations still introduce a new relation into scope from the operand to the resultant.

The ~ operator represents numeric negation, an operation that is more conventionally represented by `-`. This operator was chosen to avoid ambiguity with the infix binary subtraction operator.

3 Functions

This section lists and describes the full set of built-in functions in Quirrel. Note that built-in functions may not have user-definable semantics. A good example of this is the `count` function, which reduces a set to a single value, rather than mapping a value-level function over each of the events in that set. Built-in functions are the *only* reductions available in QUIRREL; it is not possible for users to define custom reductions.

3.1 `count`

Takes a single set and returns its size.

3.2 `load`

Takes a single set of string paths and returns the set of events in the data store which correspond to any one of the paths in question.

3.3 `max`

Takes a single set of numeric events and returns the maximum value among all the events.

3.4 `mean`

Takes a single set of numeric events and returns the arithmetic mean of all the values.

3.5 `median`

Takes a single set of numeric events and returns the median of all the values.

3.6 `min`

Takes a single set of numeric events and returns the minimum value among all the events.

3.7 `mode`

Takes a single set of numeric events and returns the mode of all the values.

3.8 `stdDev`

Takes a single set of numeric events and returns the standard deviation from the arithmetic mean of all the values.

3.9 `sum`

Takes a single set of numeric events and returns the sum of all the values.

4 Operations

This section lists and describes the full set of built-in binary operations in QUIRREL. The application semantics of these operations are defined in Section 2.9.

4.1 `where`

A pair-wise filter function where the left operand can be any related set and the right operand must be a set of boolean events. The result will be a set of events drawn from the left operand set such that the related event in the right operand is `true`. Events for which the related boolean event is `false` will be omitted from the result set.

4.2 `with`

A pair-wise conjunction operation on objects. The result for each pair of objects in the target sets will be the effective union of the sets of field/value mappings in each object, biased toward the right. Thus, if a field is present in both the left and right objects, the right definition will dominate. For pairs of values where one or both are *not* objects, the result will be undefined and will lack a corresponding value in the result set.

5 Type System

Typographical conventions:

- Γ – The lexical scope at a particular expression
- T – An expression type
- X – A type variable (assumed unique)
- e – An expression
- x – An expression variable
- p – The parameter to a characteristic function (a tic variable)
- C – A set of constraints of the form c
- c – A single type constraint

We have the following grammar for types, provenance, function types and constraints:

$$
\begin{array}{rcl}
T & ::= & \{p_1 : T_1, p_2 : T_2, \ldots, p_n : T_n\} \\
& | & [T'] \\
& | & \mathsf{Num} \\
& | & \mathsf{Bool} \\
& | & \mathsf{Str} \\
& | & \mathsf{Het} \\
& | & X \\
\\
P & ::= & v \\
& | & \forall_P\ P \\
& | & \exists_P\ P \\
& | & P_1 \cup P_2 \\
\\
F & ::= & T_1 \times T_2 \times \ldots \times T_n \to T \\
\\
c & ::= & T_1 = T_2 \\
& | & T_1 \supseteq T_2 \\
& | & T \supseteq T_1 \cup T_2
\end{array}
$$

Constraints are *definite* unless the $\supseteq$ constraint class is employed. Superset constraints unify in case of conflict (sometimes resulting in the Het type) while definite constraints error in case of conflict.

For a query to be well-typed, it must be valid according to both the provenance rules and the constraint-typing rules. In general, the following must hold:

$$\frac{\{\}, \{\} \vdash e \triangleright P \qquad \{\} \vdash e : T \mid C \qquad \sigma = \mathsf{unify}(C)}{e : \sigma(T)}$$

5.1 Values

$$\frac{(x \mapsto e) \in \Gamma \qquad e \text{ is a value}}{\Gamma \vdash x \text{ is a value}}\ \text{V-VAR} \qquad \frac{}{\Gamma \vdash {}'x \text{ is a value}}\ \text{V-TICVAR}$$

$$\frac{}{\Gamma \vdash \text{PATH-LITERAL is a value}}\ \text{V-PATH} \qquad \frac{}{\Gamma \vdash \text{STRING-LITERAL is a value}}\ \text{V-STRING}$$

$$\frac{}{\Gamma \vdash \text{NUM-LITERAL is a value}}\ \text{V-NUM} \qquad \frac{}{\Gamma \vdash \text{BOOL-LITERAL is a value}}\ \text{V-BOOL}$$

$$\frac{x \text{ is a built-in reduce function (Section 3)}}{x\texttt{(}\ldots\texttt{)} \text{ is a value}}\ \text{V-DISPATCH}$$

5.2 Provenance

In the following rules, $\mathcal{A}$ is the (possibly empty) assumption set under which provenance will be computed, while $\mathcal{P}$ is the mapping set of index to parameter id for a particular function (in Γ). We will assume the existence of a splitConj function, defined in the following way:

$$\begin{aligned} \mathsf{splitConj}(e_1 \;\&\; e_2) &= \mathsf{splitConj}(e_1) \cup \mathsf{splitConj}(e_2) \\ \mathsf{splitConj}(e) &= \{e\} \end{aligned}$$

That is, a function of an expression which produces a set of expressions consisting of the separated top-level conjunction terms. We will also assume the existence of a possibilities function defined on provenance in the following way:

$$\begin{aligned} \mathsf{possibilities}(P_1 \cup P_2) &= \{P_1 \cup P_2\} \cup \mathsf{possibilities}(P_1) \cup \mathsf{possibilities}(P_2) \\ \mathsf{possibilities}(P) &= \{P\} \end{aligned}$$

That is, a function of a provenance which returns all possible provenances which will unify with that provenance. As the union provenance is the only deconstructed case, this can also be seen as a function that enumerates all unioned provenances.

$$\frac{\begin{array}{c} \{\}, \{p_1, p_2, \ldots, p_n\}, \{\} \vdash \mathsf{crit}(e_1) = S \qquad \{e \mid S \in S' \wedge e \in \mathsf{splitConj}(e)\} = S' \\ \{p_i \mapsto P \mid e \in S' \wedge p_i \in \mathsf{vset}(e) \wedge \Gamma, \mathcal{A} \vdash e \rhd P\} = \mathcal{A}' \\ \min(\{i \mid (p_i \mapsto P) \notin \mathcal{A}'\}) = n' \\ \{i \mapsto p_i \mid i \in [1, n]\} = \mathcal{P} \qquad \Gamma \cup \{x \mapsto (\Gamma, \mathcal{A}', \mathcal{P}, e_1, n', n)\}, \mathcal{A} \vdash e_2 \rhd P \end{array}}{\Gamma, \mathcal{A} \vdash x(p_1, p_2, \ldots, p_n) \;\texttt{:=}\; e_1 \; e_2 \rhd P} \;\textsc{P-CharParam}$$

$$\frac{\Gamma \cup \{x \mapsto (\Gamma, \{\}, \{\}, e_1, 0, 0)\}, \mathcal{A} \vdash e_2 \rhd P}{\Gamma, \mathcal{A} \vdash x \;\texttt{:=}\; e_1 \; e_2 \rhd P} \;\textsc{P-Char}$$

$$\frac{\Gamma, \mathcal{A} \vdash e \rhd P'}{\Gamma, \mathcal{A} \vdash \texttt{new}\; e \rhd \exists_P \; P} \;\textsc{P-New}$$

$$\frac{\begin{array}{c} \Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \neg(\Gamma, \mathcal{A} \vdash P_1 \bowtie P_2 = P') \\ \Gamma \cup \{(P_1, P_2), (P_2, P_1)\}, \mathcal{A} \vdash e_3 \rhd P \qquad P \notin \mathsf{possibilities}(P_1) \cup \mathsf{possibilities}(P_2) \end{array}}{\Gamma, \mathcal{A} \vdash e_1 \sim e_2 \; e_3 \rhd P} \;\textsc{P-RelateNoop}$$

$$\frac{\begin{array}{c} \Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \neg(\Gamma \vdash P_1 \bowtie P_2 = P') \\ \Gamma \cup \{(P_1, P_2), (P_2, P_1)\}, \mathcal{A} \vdash e_3 \rhd P \qquad P \in \mathsf{possibilities}(P_1) \cup \mathsf{possibilities}(P_2) \end{array}}{\Gamma, \mathcal{A} \vdash e_1 \sim e_2 \; e_3 \rhd \exists_P \; P} \;\textsc{P-Relate}$$

$$\frac{\begin{array}{c}(x \mapsto (\Gamma', \mathcal{A}', \mathcal{P}, e', n', m)) \in \Gamma \qquad n \geq n' \qquad n < m \qquad \Gamma, \mathcal{A} \vdash e_i \triangleright P_i \\ \Gamma \vdash P_1 \bowtie P_2 \bowtie \ldots \bowtie P_n = P_\cup \qquad \{p_i \mapsto P_i \mid i \in [1,n] \wedge (i \mapsto p_i) \in \mathcal{P}\} = \mathcal{A}_2 \\ (\mathcal{A}' \setminus \{p_k \mapsto P_k \mid (p_k \mapsto P_k) \in \mathcal{A}' \wedge k \leq n\}) \cup \mathcal{A}_2 = \mathcal{A}'' \\ \Gamma', \mathcal{A}'' \vdash e' \triangleright P'\end{array}}{\Gamma, \mathcal{A} \vdash x\texttt{(}e_1, e_2, \ldots, e_n\texttt{)} \triangleright \exists_P\ P} \text{ P-DispatchPartial}$$

$$\frac{\begin{array}{c}(x \mapsto (\Gamma', \mathcal{A}', \mathcal{P}, e', n', n)) \in \Gamma \qquad \Gamma, \mathcal{A} \vdash e_i \triangleright P_i \\ \Gamma \vdash P_1 \bowtie P_2 \bowtie \ldots \bowtie P_n = P \qquad \{p_i \mapsto P_i \mid i \in [1,n] \wedge (i \mapsto p_i) \in \mathcal{P}\} = \mathcal{A}_2 \\ (\mathcal{A}' \setminus \{p_k \mapsto P_k \mid (p_k \mapsto P_k) \in \mathcal{A}' \wedge k \leq n\}) \cup \mathcal{A}_2 = \mathcal{A}'' \qquad \Gamma', \mathcal{A}'' \vdash e' \triangleright P'\end{array}}{\Gamma, \mathcal{A} \vdash x\texttt{(}e_1, e_2, \ldots, e_n\texttt{)} \triangleright P} \text{ P-DispatchTotal}$$

$$\frac{\begin{array}{c}(\texttt{load} \mapsto (\Gamma', \mathcal{A}', \mathcal{P}, e', n', m)) \notin \Gamma \\ \Gamma, \mathcal{A} \vdash e \text{ is a value} \qquad e \Rightarrow v \qquad \Gamma, \mathcal{A} \vdash e \triangleright P'\end{array}}{\Gamma, \mathcal{A} \vdash \texttt{load(}e\texttt{)} \triangleright v} \text{ P-LoadValue}$$

$$\frac{\begin{array}{c}(\texttt{load} \mapsto (\Gamma', \mathcal{A}', \mathcal{P}, e', n', m)) \notin \Gamma \\ \neg(\Gamma, \mathcal{A} \vdash e \text{ is a value}) \qquad \Gamma, \mathcal{A} \vdash e \triangleright P'\end{array}}{\Gamma, \mathcal{A} \vdash \texttt{load(}e\texttt{)} \triangleright \exists_P\ P} \text{ P-LoadDynamic}$$

$$\frac{\begin{array}{c}(x \mapsto (\Gamma', \mathcal{A}', \mathcal{P}, e', n', m)) \notin \Gamma \\ x \in \text{built in functions (Section 3)} \\ \Gamma, \mathcal{A} \vdash e_i \triangleright P_i \qquad \mathcal{A} = \{\}\end{array}}{\Gamma, \mathcal{A} \vdash x(e_1, e_2, \ldots, e_n) \triangleright \forall_P\ P} \text{ P-DispatchBuiltInUnassuming}$$

$$\frac{\begin{array}{c}(x \mapsto (\Gamma', \mathcal{A}', \mathcal{P}, e', n', m)) \notin \Gamma \\ x \in \text{built in functions (Section 3)} \qquad \mathcal{A} \neq \{\}\end{array}}{\Gamma, \mathcal{A} \vdash x(e_1, e_2, \ldots, e_n) \triangleright \forall_P\ P} \text{ P-DispatchBuiltInAssuming}$$

$$\frac{('x \mapsto P) \in \mathcal{A}}{\Gamma, \mathcal{A} \vdash {'x} \triangleright P} \text{ P-TicVarAssumed} \qquad \frac{('x \mapsto P') \notin \mathcal{A}}{\Gamma, \mathcal{A} \vdash {'x} \triangleright \forall_P\ P} \text{ P-TicVar}$$

$$\frac{}{\Gamma, \mathcal{A} \vdash \text{Path-Literal} \triangleright \forall_P\ P} \text{ P-Path} \qquad \frac{}{\Gamma, \mathcal{A} \vdash \text{String-Literal} \triangleright \forall_P\ P} \text{ P-String}$$

$$\frac{}{\Gamma, \mathcal{A} \vdash \text{Num-Literal} \triangleright \forall_P\ P} \text{ P-Num} \qquad \frac{}{\Gamma, \mathcal{A} \vdash \text{Bool-Literal} \triangleright \forall_P\ P} \text{ P-Bool}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_i \rhd P_i \qquad \Gamma \vdash P_1 \bowtie P_2 \bowtie \ldots \bowtie P_n = P}{\Gamma, \mathcal{A} \vdash \{p_1 : e_1, p_2 : e_2, \ldots, p_n : e_n\} \rhd P} \text{ P-}\textsc{Object}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_i \rhd P_i \qquad \Gamma \vdash P_1 \bowtie P_2 \bowtie \ldots \bowtie P_n = P}{\Gamma, \mathcal{A} \vdash \texttt{[}e_1, e_2, \ldots, e_n\texttt{]} \rhd P} \text{ P-}\textsc{Array}$$

$$\frac{\Gamma, \mathcal{A} \vdash e \rhd P}{\Gamma, \mathcal{A} \vdash e\,\texttt{.}\,p \rhd P} \text{ P-}\textsc{Descent} \qquad \frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1\,\texttt{[}e_2\texttt{]} \rhd P} \text{ P-}\textsc{Deref}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; x \; e_2 \rhd P} \text{ P-}\textsc{Operation}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{+} \; e_2 \rhd P} \text{ P-}\textsc{Add}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{-} \; e_2 \rhd P} \text{ P-}\textsc{Sub}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{*} \; e_2 \rhd P} \text{ P-}\textsc{Mul}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{/} \; e_2 \rhd P} \text{ P-}\textsc{Div}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{<} \; e_2 \rhd P} \text{ P-}\textsc{Lt}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{<=} \; e_2 \rhd P} \text{ P-}\textsc{LtEq}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{>} \; e_2 \rhd P} \text{ P-}\textsc{Gt}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \rhd P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \rhd P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \; \texttt{>=} \; e_2 \rhd P} \text{ P-}\textsc{GtEq}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \triangleright P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \triangleright P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \texttt{ = } e_2 \triangleright P} \text{ P-E}\textsc{q}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \triangleright P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \triangleright P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \texttt{ != } e_2 \triangleright P} \text{ P-N}\textsc{ot}\text{E}\textsc{q}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \triangleright P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \triangleright P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \texttt{ \& } e_2 \triangleright P} \text{ P-A}\textsc{nd}$$

$$\frac{\Gamma, \mathcal{A} \vdash e_1 \triangleright P_1 \qquad \Gamma, \mathcal{A} \vdash e_2 \triangleright P_2 \qquad \Gamma \vdash P_1 \bowtie P_2 = P}{\Gamma, \mathcal{A} \vdash e_1 \texttt{ | } e_2 \triangleright P} \text{ P-O}\textsc{r}$$

$$\frac{\Gamma, \mathcal{A} \vdash e \triangleright P}{\Gamma, \mathcal{A} \vdash \texttt{!}e \triangleright P} \text{ P-C}\textsc{omp} \qquad\qquad \frac{\Gamma, \mathcal{A} \vdash e \triangleright P}{\Gamma, \mathcal{A} \vdash \texttt{neg}\ e \triangleright P} \text{ P-N}\textsc{eg}$$

$$\frac{\Gamma, \mathcal{A} \vdash e \triangleright P}{\Gamma, \mathcal{A} \vdash \texttt{(}e\texttt{)} \triangleright P} \text{ P-P}\textsc{aren}$$

5.2.1 Unification

$$\frac{}{\Gamma \vdash P \bowtie \forall_P\ P = P} \text{ PU-V}\textsc{alue}\text{1} \qquad\qquad \frac{}{\Gamma \vdash \forall_P\ P \bowtie P = P} \text{ PU-V}\textsc{alue}\text{2}$$

$$\frac{}{\Gamma \vdash v \bowtie v = v} \text{ PU-S}\textsc{tatic} \qquad\qquad \frac{\Gamma \vdash P_1 \bowtie P = P'}{\Gamma \vdash P_1 \cup P_2 \bowtie P = P'} \text{ PU-U}\textsc{nion}\text{1}$$

$$\frac{\Gamma \vdash P_2 \bowtie P = P'}{\Gamma \vdash P_1 \cup P_2 \bowtie P = P'} \text{ PU-U}\textsc{nion}\text{2} \qquad\qquad \frac{\Gamma \vdash P_1 \bowtie P = P_1' \qquad \Gamma \vdash P_2 \bowtie P = P_2'}{\Gamma \vdash P_1 \cup P_2 \bowtie P = P_1' \cup P_2'} \text{ PU-U}\textsc{nion}$$

$$\frac{}{\Gamma \vdash \exists_P\ (P \bowtie P) = \exists_P\ P} \text{ PU-D}\textsc{ynamic} \qquad\qquad \frac{(P_1, P_2) \in \Gamma}{\Gamma \vdash P_1 \bowtie P_2 = P_1 \cup P_2} \text{ PU-E}\textsc{xplicit}$$

$$\frac{\Gamma \vdash P_1 \bowtie P_2 = P' \qquad (P_2, P_3) \in \Gamma}{\Gamma \vdash P_1 \bowtie P_3 = P' \cup P_3} \text{ PU-T}\textsc{ransitive}$$

5.2.2 Critical Conditions

In the inference rules below, $\mathcal{P}$ represents the set of parameters (assumed unique) for which we are defining the critical conditions, while $\mathcal{W}$ represents the (possibly empty) singleton set of the currently active predicate. We also assume the existence of a merge function defined in the following way:

$$\mathsf{merge}(S_1, S_2) = \{p \mapsto S'_1 \cup S'_2 \mid (p \mapsto S'_1) \in S_1 \land (p \mapsto S'_2) \in S_2\}$$

That is, a function of two sets of mappings where the mapping results are themselves sets and returns a single set of mappings where the keys are mapped to the union of their value sets in the given sets of mappings.

Note that critical conditions are a step towards structurally finding some set $\mathcal{S}'(F) \supseteq \mathcal{S}(F)$, where $\mathcal{S}(F)$ is the sub-domain of the characteristic function F. That is to say that critical conditions are a step toward approximating the sub-domain of a function. See Section 6.1.

$$\frac{\Gamma \cup \{x \mapsto (\Gamma, e_1)\}, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(x\,\texttt{(}p_1, p_2, \ldots, p_n\texttt{)}\ \texttt{:=}\ e_1\ e_2) = S}\ \textsc{Crit-CharParam}$$

$$\frac{\Gamma \cup \{x \mapsto (\Gamma, e_1)\}, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(x\ \texttt{:=}\ e_1\ e_2) = S}\ \textsc{Crit-CharParam}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\texttt{new}\ e) = S}\ \textsc{Crit-New}$$

$$\frac{\begin{array}{c} \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \\ \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_3) = S_3 \qquad \mathsf{merge}(S_1, S_2, S_3) = S \end{array}}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \sim e_2\ e_3) = S}\ \textsc{Crit-Relate}$$

$$\frac{\begin{array}{c} (x \mapsto (\Gamma', e')) \in \Gamma \qquad \Gamma', \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e') = S' \\ \mathsf{merge}(S', \mathsf{merge}(\{S_i \mid \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_i) = S_i\})) = S \end{array}}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(x\,\texttt{(}e_1, e_2, \ldots, e_n\texttt{)}) = S}\ \textsc{Crit-BoundDispatchParam}$$

$$\frac{(x \mapsto (\Gamma', e')) \in \Gamma \qquad \Gamma', \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e') = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(x) = S}\ \textsc{Crit-BoundDispatchNull}$$

$$\frac{\begin{array}{c} (x \mapsto (\Gamma', e')) \notin \Gamma \\ \mathsf{merge}(\{S_i \mid \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_i) = S_i\}) = S \end{array}}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(x\,\texttt{(}e_1, e_2, \ldots, e_n\texttt{)}) = S}\ \textsc{Crit-UnBoundDispatchParam}$$

$$\frac{(x \mapsto (\Gamma', e')) \notin \Gamma}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(x) = \{\}}\ \textsc{Crit-UnBoundDispatchNull}$$

$$\frac{'x \in \mathcal{P}}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}('x) = \{'x \mapsto \mathcal{W}\}}\ \textsc{Crit-TicVarTarget} \qquad \frac{'x \notin \mathcal{P}}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}('x) = \{\}}\ \textsc{Crit-TicVar}$$

$$\frac{}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\textsc{Path-Literal}) = \{\}}\ \textsc{Crit-Path}$$

$$\frac{}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\textsc{String-Literal}) = \{\}}\ \textsc{Crit-Str}$$

$$\frac{}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\textsc{Num-Literal}) = \{\}}\ \textsc{Crit-Num}$$

$$\frac{}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\textsc{Bool-Literal}) = \{\}}\ \textsc{Crit-Bool}$$

$$\frac{\mathsf{merge}(\{S_i \mid \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_i) = S_i\}) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\{p_1 : e_1, p_2 : e_2, \ldots, p_n : e_n\}) = S}\ \textsc{Crit-Object}$$

$$\frac{\mathsf{merge}(\{S_i \mid \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_i) = S_i\}) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\texttt{[}e_1, e_2, \ldots, e_n\}) = S}\ \textsc{Crit-Array}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e\,\texttt{.}\,p) = S}\ \textsc{Crit-Descent}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1\texttt{[}e_2\texttt{]}) = S}\ \textsc{Crit-Deref}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \{e_2\} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1\ \texttt{where}\ e_2) = S}\ \textsc{Crit-Where}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ + } e_2) = S} \textsc{ Crit-Add}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ - } e_2) = S} \textsc{ Crit-Sub}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ * } e_2) = S} \textsc{ Crit-Mul}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ / } e_2) = S} \textsc{ Crit-Div}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ < } e_2) = S} \textsc{ Crit-Lt}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ <= } e_2) = S} \textsc{ Crit-LtEq}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ > } e_2) = S} \textsc{ Crit-Gt}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ >= } e_2) = S} \textsc{ Crit-GtEq}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ = } e_2) = S} \textsc{ Crit-Eq}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ != } e_2) = S} \textsc{ Crit-NotEq}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ \& } e_2) = S} \textsc{ Crit-And}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1) = S_1 \qquad \Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_2) = S_2 \qquad \mathsf{merge}(S_1, S_2) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e_1 \texttt{ | } e_2) = S} \textsc{ Crit-Or}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\texttt{!}e) = S} \textsc{ Crit-Comp} \qquad\qquad \frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\texttt{neg}\ e) = S} \textsc{ Crit-Neg}$$

$$\frac{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(e) = S}{\Gamma, \mathcal{P}, \mathcal{W} \vdash \mathsf{crit}(\texttt{(}e\texttt{)}) = S}\ \textsc{Crit-Paren}$$

5.2.3 Variable Sets

$$\frac{\mathsf{vset}(e_1) = S_1 \qquad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(x\texttt{(}p_1, p_2, \ldots, p_n\texttt{)}\ \texttt{:=}\ e_1\ e_2) = S_1 \cup S_2}\ \text{VS-}\textsc{CharParam}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \qquad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(x\ \texttt{:=}\ e_1\ e_2) = S_1 \cup S_2}\ \text{VS-}\textsc{CharParam}$$

$$\frac{}{\mathsf{vset}(\texttt{new}\ e) = \mathsf{vset}(e)}\ \text{VS-}\textsc{New}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \qquad \mathsf{vset}(e_2) = S_2 \qquad \mathsf{vset}(e_3) = S_3}{\mathsf{vset}(e_1 \sim e_2\ e_3) = S_1 \cup S_2 \cup S_3}\ \text{VS-}\textsc{Relate}$$

$$\frac{}{\mathsf{vset}(x\texttt{(}e_1, e_2, \ldots, e_n\texttt{)}) = \bigcup_{i \leq n} \mathsf{vset}(e_i)}\ \text{VS-}\textsc{DispatchParam}$$

$$\frac{}{\mathsf{vset}(x) = \{\}}\ \text{VS-}\textsc{DispatchNull}$$

$$\frac{}{\mathsf{vset}('x) = \{'x\}}\ \text{VS-}\textsc{TicVar}$$

$$\frac{}{\mathsf{vset}(\textsc{Path-Literal}) = \{\}}\ \text{VS-}\textsc{Path}$$

$$\frac{}{\mathsf{vset}(\textsc{String-Literal}) = \{\}}\ \text{VS-}\textsc{Str}$$

$$\frac{}{\mathsf{vset}(\textsc{Num-Literal}) = \{\}}\ \text{VS-}\textsc{Num}$$

$$\frac{}{\mathsf{vset}(\textsc{Bool-Literal}) = \{\}}\ \text{VS-}\textsc{Bool}$$

$$\frac{}{\mathsf{vset}(\{p_1 \colon e_1, p_2 \colon e_2, \ldots, p_n \colon e_n\}) = \bigcup_{i \leq n} \mathsf{vset}(e_i)}\ \text{VS-}\textsc{Object}$$

$$\frac{}{\mathsf{vset}(\texttt{[}e_1, e_2, \ldots, e_n\texttt{]}) = \bigcup_{i \leq n} \mathsf{vset}(e_i)}\ \text{VS-}\textsc{Array}$$

$$\frac{}{\mathsf{vset}(e\,\texttt{.}p) = \mathsf{vset}(e)}\ \textsc{VS-Descent} \qquad \frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\,\texttt{[}e_2\texttt{]}) = S_1 \cup S_2}\ \textsc{VS-Deref}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{where}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Where}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{+}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Add} \qquad \frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{-}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Sub}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{*}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Mul} \qquad \frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{/}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Div}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{<}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Lt} \qquad \frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{<=}\ e_2) = S_1 \cup S_2}\ \textsc{VS-LtEq}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{>}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Gt} \qquad \frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{>=}\ e_2) = S_1 \cup S_2}\ \textsc{VS-GtEq}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{=}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Eq} \qquad \frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{!=}\ e_2) = S_1 \cup S_2}\ \textsc{VS-NotEq}$$

$$\frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{\&}\ e_2) = S_1 \cup S_2}\ \textsc{VS-And} \qquad \frac{\mathsf{vset}(e_1) = S_1 \quad \mathsf{vset}(e_2) = S_2}{\mathsf{vset}(e_1\ \texttt{|}\ e_2) = S_1 \cup S_2}\ \textsc{VS-Or}$$

$$\frac{}{\mathsf{vset}(\texttt{!}e) = \mathsf{vset}(e)}\ \textsc{VS-Comp} \qquad \frac{}{\mathsf{vset}(\texttt{neg}\ e) = \mathsf{vset}(e)}\ \textsc{VS-Neg}$$

$$\frac{}{\mathsf{vset}(\texttt{(}e\texttt{)}) = \mathsf{vset}(e)}\ \textsc{VS-Paren}$$

5.3 Constraint Typing

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \cup \{x : T_1, p_1, p_2, \ldots, p_n\} \vdash e_2 : T_2 \mid C}{\Gamma \vdash x(p_1, p_2, \ldots, p_n)\,\texttt{:=}e_1\ e_2 : T_2 \mid C} \text{ CT-CharParam}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \cup \{x : T_1\} \vdash e_2 : T_2 \mid C}{\Gamma \vdash x\,\texttt{:=}e_1\ e_2 : T_2 \mid C} \text{ CT-Char}$$

$$\frac{\Gamma \vdash e : T \mid C}{\Gamma \vdash \texttt{new}\ e : T \mid C} \text{ CT-New} \qquad \frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad \Gamma \vdash e_3 : T_3 \mid C}{\Gamma \vdash e_1 \sim e_2\ e_3 : T_3 \mid C} \text{ CT-Relate}$$

$$\frac{\forall_{i \in [0,n]}\ \Gamma \vdash e_i : T_i \mid C \qquad (x : T) \in \Gamma}{\Gamma \vdash x\texttt{(}e_1, e_2, \ldots, e_n\texttt{)} : X \mid C \cup \{X = T\}} \text{ CT-DispatchParam}$$

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : X \mid C \cup \{X = T\}} \text{ CT-DispatchNull}$$

$$\frac{\begin{array}{c}(x : T') \notin \Gamma \\ (x : T_1 \times T_2 \times \ldots \times T_n \to T) \in \text{ built in functions (Section 3)} \\ \forall_{i \in [0,n]}\ e_i : T'_i \mid C \qquad C \cup \bigcup_i \{T'_i \supseteq T_i\} \cup \{X = T\} = C'\end{array}}{\Gamma \vdash x\texttt{(}e_1, e_2, \ldots, e_n\texttt{)} : X \mid C'} \text{ CT-DispatchBuiltIn}$$

$$\frac{\Gamma \vdash e : T \mid C}{\Gamma \vdash \texttt{load(}e\texttt{)} : X \mid C \cup \{T \supseteq \mathsf{Str}\}} \text{ CT-Load}$$

$$\frac{'x \in \Gamma}{\Gamma \vdash' x : X \mid \{\}} \text{ CT-TicVar} \qquad \frac{}{\Gamma \vdash \text{Path-Literal} : X \mid \{X = \mathsf{Str}\}} \text{ CT-Path}$$

$$\frac{}{\Gamma \vdash \text{String-Literal} : X \mid \{X = \mathsf{Str}\}} \text{ CT-Str}$$

$$\frac{}{\Gamma \vdash \text{Num-Literal} : X \mid \{X = \mathsf{Num}\}} \text{ CT-Num}$$

$$\frac{}{\Gamma \vdash \text{Bool-Literal} : X \mid \{X = \mathsf{Bool}\}} \text{ CT-Bool}$$

$$\frac{\forall_{i\in[0,n]}\ e_i : T_i \mid C \qquad T = \bigcup_i \{p_i : T_i\} \qquad C \cup \{X = T\} = C'}{\Gamma \vdash \{p_1 : e_1, p_2 : e_2, \ldots, p_n : e_n\} : X \mid C'}\ \text{CT-Object}$$

$$\frac{\forall_{i\in[0,n]}\ e_i : T \mid C \qquad C \cup \{X = [T]\} = C'}{\Gamma \vdash \texttt{[}e_1, e_2, \ldots, e_n\texttt{]} : X \mid C'}\ \text{CT-ArrayHomo}$$

$$\frac{\forall_{i\in[0,n]}\ e_i : T_i \mid C \qquad \exists_{i\in[0,n]}\exists_{j\in[0,n]}\ T_i \neq T_j \qquad C \cup \{X = [\mathsf{Het}]\} = C'}{\Gamma \vdash \texttt{[}e_1, e_2, \ldots, e_n\texttt{]} : X \mid C'}\ \text{CT-ArrayHetero}$$

$$\frac{\Gamma \vdash e : T \mid C \qquad C \cup \{T \supseteq T' \cup \{p : X\}\} = C'}{\Gamma \vdash e\,.\,p : X \mid C'}\ \text{CT-Descent}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq [X], e_2 \supseteq \mathsf{Num}\} = C'}{\Gamma \vdash e_1\texttt{[}e_2\texttt{]} : X \mid C'}\ \text{CT-Deref}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_2 \supseteq \mathsf{Bool}\}}{\Gamma \vdash e_1\ \texttt{where}\ e_2 : T_1 \mid C'}\ \text{CT-Where}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Num}\} = C'}{\Gamma \vdash e_1\ \texttt{+}\ e_2 : X \mid C'}\ \text{CT-Add}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Num}\} = C'}{\Gamma \vdash e_1\ \texttt{-}\ e_2 : X \mid C'}\ \text{CT-Sub}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Num}\} = C'}{\Gamma \vdash e_1\ \texttt{*}\ e_2 : X \mid C'}\ \text{CT-Mul}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Num}\} = C'}{\Gamma \vdash e_1\ \texttt{/}\ e_2 : X \mid C'}\ \text{CT-Div}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{<}} e_2 : X \mid C'} \ \textsc{CT-Lt}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{<=}} e_2 : X \mid C'} \ \textsc{CT-LtEq}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{>}} e_2 : X \mid C'} \ \textsc{CT-Gt}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Num}, T_2 \supseteq \mathsf{Num}, X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{>=}} e_2 : X \mid C'} \ \textsc{CT-GtEq}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{=}} e_2 : X \mid C'} \ \textsc{CT-Eq}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{=}} e_2 : X \mid C'} \ \textsc{CT-NotEq}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Bool}, T_2 \supseteq \mathsf{Bool}, X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{\&}} e_2 : X \mid C'} \ \textsc{CT-And}$$

$$\frac{\Gamma \vdash e_1 : T_1 \mid C \qquad \Gamma \vdash e_2 : T_2 \mid C \qquad C \cup \{T_1 \supseteq \mathsf{Bool}, T_2 \supseteq \mathsf{Bool}, X = \mathsf{Bool}\} = C'}{\Gamma \vdash e_1 \mathrel{\mathsf{|}} e_2 : X \mid C'} \ \textsc{CT-Or}$$

$$\frac{\Gamma \vdash e : T \mid C \qquad C \cup \{T \supseteq \mathsf{Bool}, X = \mathsf{Bool}\} = C'}{\Gamma \vdash \mathsf{!}e : X \mid C'} \ \textsc{CT-Comp}$$

$$\frac{\Gamma \vdash e : T \mid C \qquad C \cup \{T \supseteq \mathsf{Num}, X = \mathsf{Num}\} = C'}{\Gamma \vdash \mathsf{neg}\ e : X \mid C'} \ \textsc{CT-Neg}$$

$$\frac{\Gamma \vdash e : T \mid C}{\Gamma \vdash \mathsf{(}e\mathsf{)} : T \mid C} \ \textsc{CT-Paren}$$

6 Formal Meta-Theory

We will define a "dataset" to be a set of events. An event is a JSON value paired with identity. All events are of the following form:

$$e = (I, v)$$

Identity (I) is a (possibly empty) set of identity values, i_1, i_2, etc. Identity values are presumed unique. Sets may contain duplicate values if those values have distinct identity.

Sets are defined by characteristic functions parameterized by universally-quantified variables. We define characteristic functions in terms of a arity-0 or -1, but these definitions generalize naturally to higher-arity forms. Consider the following notational form:

$$S \mathrel{:=} E$$

This introduces a new set, S, containing the results produced by evaluating the expression, E. This is a characteristic function of arity-0. Now consider the following, less trivial form:

$$S(p) \mathrel{:=} E[p]$$

We are assuming the notation $E[p]$ to mean "the expression E defined in terms of free parameter p." This is a characteristic function of arity-1. It introduces not one, but *two* sets defined according to the following identities:

$$S = \bigcup_{e \in \mathcal{U}} E[e]$$

$$S(e) = E[e]$$

Thus, a set is defined as an expression which is defined in terms of some parameters. Any parameters that are left unfixed when accessing the set are universally quantified. All expressions produce sets.

For some expressions, $E[p]$, the unquantified form of S (accessed without instantiating the formal parameter) may be infinite or otherwise undefined. For example, if the expression $E[p]$ simply produces the value p, then the set S will be $\mathcal{U}$, which is not well defined. Similarly, the expression $E[p]$ may be defined according to an operation pairing p with some finite set, but uses p without imposing any constraints upon its value. In such cases, the set S will certainly be finite, but has no definite value since p may take on any values.

Thus, the only expressions $E[p]$ for which S may be applied unquantified are those for which p may only take on a finite, well defined set of values. For such sets, all values of p which are outside this finite, well defined set must cause $E[p] \rightarrow^* \emptyset$. Thus, for any well defined unquantified instantiation of a characteristic function, there will be an infinite set of values for p which yield the empty set (and are thus lost in the union), and a finite and well defined set of values for p which yield a non-empty set. We call this set of values for p the *sub-domain*, as defined below.

6.1 Sub-Domains

The domain of all characteristic functions of arity-1 is $\mathcal{U}$. This is unenlightening. We introduce the notion of a *sub-domain*, $\mathcal{S}(F)$, which is a subset of the full domain. This sub-domain is given by the following definition[1]:

$$\mathcal{S}_F = \bigcup \{e \mid F(e) \neq \emptyset\}$$

Note that the sub-domain will usually satisfy the following pair of invariants[2]:

$$\forall_{e \in \mathcal{S}_F} \; F(\mathcal{S}_F \setminus \{e\}) \subset F(\mathcal{S}_F)$$

[1]We are assuming the notational conventions defined in Section 6.2

[2]The term "invariant" is too string, since these properties are not *always* satisfied (see Section 6.1.1)

$$F(\mathcal{S}_F) = F(\mathcal{U})$$

Note that if F is an arbitrary function, then $\mathcal{S}(F)$ cannot be decided by a Turing machine. This follows from Rice's Theorem if we see the set $\mathcal{S}_F$ as defining the effective domain of the function, F. We cannot prove a distinction between $\mathcal{S}_F$ and $\mathcal{S}'_F \subset \mathcal{S}_F$ in finite time. That is to say that we cannot decide the proposition that F is total on its effective domain. For this reason, F cannot be an arbitrary Turing computable function.

Intuitively, the sub-domain represents the subset of the domain that is effective in the function. When the function is applied to values that are *not* in the sub-domain, the result is the empty set. As unquantified characteristic function application is defined according to the big union operation, function results from values that are not in the sub-domain do not contribute to the final result. The sub-domain is the largest set of values such that the function applied to any value in the set produces a non-empty result set.

6.1.1 Exceptions

Note that the sub-domain may not actually satisfy its two invariants. This will happen when $\exists_{e_1}\exists_{e_2}\ F(e_1) = F(e_2)$. Simply following the invariants to construct the sub-domain in this case will lead to more than one possible sub-domain set: one which contains e_1 and one which contains e_2. The constructive definition of $\mathcal{S}_F$ avoids ambiguity in this case by defining the sub-domain to be the maximal set of values for which the function application is non-empty. The resulting sub-domain will be uniquely defined, but it will *not* satisfy the invariants as it will be possible to remove a value (either e_1 or e_2 from the earlier expression) without reducing the unified result set.

An unfortunate consequence of this resolution of the ambiguity is the fact that not all well-defined functions have a finite sub-domain. A trivial example of this would be the constant function:

$$\mathsf{const}(p) \mathrel{:=} 42$$

The const function is defined for all values of p, but it always returns the same (non-empty) set for any given value. Thus, the sub-domain is $\mathcal{U}$ according to the unambiguous constructive definition. Conversely, if we build the sub-domain according to the invariants, any singleton set would suffice.

Fortunately, these distinctions are irrelevant for both practical and theoretical considerations. Practically, an implementation may simply choose *any* of the possible sub-domain sets that obey the invariants. Well-defined functions will always have at least one sub-domain that obeys the invariants and is finite. For theoretical work, either a guaranteed-finite or potentially-infinite sub-domain may be acceptable, depending on the problem. The only problems which cannot be examined are those which require a guaranteed-finite sub-domain that is also guaranteed to be unique. Unfortunately, as the const example shows, there can be no definitions of the sub-domain which yield such properties, and thus such questions are inherently unanswerable without reduction to one of the tractable sub-domain cases.

6.2 Operations

All value operations are implicitly lifted to map over sets of events. This lifting is defined in the following way for unary operations:

$$\begin{aligned}&\text{let } f : v \to v\\ \\ &F(S) = \{(I, f(v)) \mid (I, v) \in S\}\\ &R_{S-F(S)} = \{((I, v), (I, f(v))) \mid (I, v) \in S\}\end{aligned}$$

For every unary function on values, f, we also have a unary function on sets, F, which maps f over all the values in a given set. We also have a relation, R, from S to $F(S)$. See Section 6.4.

We can define an analogous lifting for binary functions:

$$\text{let } f : v \times v \to v$$

$$F(S_1, S_2) = \left\{ (I_1 \cup I_2, f(v_1, v_2)) \;\middle|\; \begin{array}{l} (I_1, v_1) \in S_1 \\ (I_2, v_2) \in S_2 \\ ((I_1, v_1), (I_2, v_2)) \in \bigcap R_{S_1 - S_2} \end{array} \right\}$$

$$R_{S_1-F(S_1,S_2)} = \left\{ ((I_1, v_1), (I_1 \cup I_2, f(v_1, v_2))) \;\middle|\; \begin{array}{l} (I_1, v_1) \in S_1 \\ (I_2, v_2) \in S_2 \\ ((I_1, v_1), (I_2, v_2)) \in \bigcap R_{S_1 - S_2} \end{array} \right\}$$

$$R_{S_2-F(S_1,S_2)} = \left\{ ((I_2, v_2), (I_1 \cup I_2, f(v_1, v_2))) \;\middle|\; \begin{array}{l} (I_1, v_1) \in S_1 \\ (I_2, v_2) \in S_2 \\ ((I_1, v_1), (I_2, v_2)) \in \bigcap R_{S_1 - S_2} \end{array} \right\}$$

Thus, for any binary function on values, f, we also have a binary function on sets, F, which maps f over all pairs of values in the sets such that there is a relation between the two sets that contains the pair of events. The notation fragment $\bigcap R_{S_1-S_2}$ hints at the fact that multiple relations may exist for a pair of sets, S_1 and S_2. In such cases, we consider the intersection of all such relations.

There is additionally a special operation, new, that takes a set and returns a set containing the same values with fresh identities:

$$\mathsf{new}(S) = \{(\{i'\}, v) \mid \exists_{i'}\ (I, v) \in S \land (\{i'\}, v) \notin S\} \text{ such that } \forall_{(I,v)\in\mathsf{new}(\mathsf{new}(S))}\ (I, v) \notin S$$

6.3 Lifting

For any value, we have an equivalent event with the empty identity. This lifting is handled implicitly by the lift function as defined:

$$\mathsf{lift}(v) = (\{\}, v)$$

Additionally, for any event, we have the singleton set consisting of just that event. This lifting is handled implicitly by the Lift function (which may be implicitly chained with lift):

$$\begin{aligned}&\mathsf{Lift}(e) = \{e\}\\ &\forall_S\ R_{S-\mathsf{Lift}(e)} = \{(e', e) \mid e' \in S\}\end{aligned}$$

Thus, any event lifting also introduces a relation from *any* set to the lifted singleton set where all events in the set are paired with the single event which was lifted.

Unfortunately, this encoding is somewhat problematic when taken in conjunction with the R-Trans rule given in Section 6.4. Thus, a more principled approach is required:

$$\begin{array}{l}\mathsf{Lift}(e) = \{e\} \\ \lambda_S \; R_{S-\mathsf{Lift}(e)} = \{(e', e) \mid e' \in S\}\end{array}$$

Thus, any implicit lifting must be applied at the point of reference to a specific set. When the lifting occurs, a relation is generated *for that set* to the lifted singleton set.

6.4 Relations

Relations between sets are used to determine which events from one set align with what events from the other set in lifted binary operations. All relations are a subset of the following (for some sets S_1 and S_2):

$$R_{S_1-S_2} = \{(e_1, e_2) \mid e_1 \in S_1 \wedge e_2 \in S_2\}$$

Relations are introduced by lifted operations, both unary and binary, lifted values (see Section 6.3), and by the following set of rules:

$$\frac{}{R_{S-S} = \{(e, e) \mid e \in S\}}\ \textsc{R-Refl} \qquad \frac{}{R_{S_1-S_3} = \{(e_1, e_3) \mid (e_1, e_2) \in R_{S_1-S_2} \wedge (e_2, e_3) \in R_{S_2-S_3}\}}\ \textsc{R-Trans}$$

$$\frac{}{R_{S_2-S_1} = \{(e_2, e_1) \mid (e_1, e_2) \in R_{S_1-S_2}\}}\ \textsc{R-Commute} \qquad \frac{S_1 \sim S_2 \qquad \neg\exists\, R_{S_1-S_2}}{R_{S_1-S_2} = S_1 \times S_2}\ \textsc{R-Def}$$

We can track the existence of relations implicitly by introducing the notion of set *provenance*. If two sets have the same provenance, then there must exist a unique minimal relation between them. This mechanism greatly simplifies the amount of bookkeeping required to check the relational validity of expressions. The provenance system is specified in Section 5.2. Compositional semantics for the provenance system are specified in Section 5.2.1.

Glossary

event A single value paired with an identity . 5–7

fixity Describes the syntactic positioning of an operation with respect to its operand(s) . 29

identity A unique value internal to the runtime, or bottom ($\perp$) . 6

infix Describes an operation which is syntactically positioned *between* its operands . 7, 29

mixfix Describes an operation with a fixity that is a combination of prefix, postfix and infix . 7

nary Describes a function or operation that accepts an unfixed number of operands . 7

postfix Describes an operation which is syntactically positioned *after* to its operand(s) . 29

prefix Describes an operation which is syntactically positioned *prior* to its operand(s) . 29

relation A pair-wise relation between two sets such that one or more elements from the first set are related to one or more elements from the second set . 6

set A mathematical set of events . 5, 6

tic variable A formal parameter for some characteristic function . 5, 6, 11

value A single JSON value such as a number, boolean, string, array or object . 5, 6