On Generalized Records and Spatial Conjunction in Role Logic

Viktor Kuncak and Martin Rinard

MIT Computer Science and Artificial Intelligence Laboratory
{vkuncak, rinard}@csail.mit.edu

MIT CSAIL Technical Report No 942 VK0104

Abstract. We have previously introduced role logic as a notation for describing properties of relational structures in shape analysis, databases and knowledge bases. A natural fragment of role logic corresponds to two-variable logic with counting and is therefore decidable.
We show how to use role logic to describe open and closed records, as well the dual of records, inverse records. We observe that the spatial conjunction operation of separation logic naturally models record concatenation. Moreover, we show how to eliminate the spatial conjunction of formulas of quantifier depth one in first-order logic with counting. As a result, allowing spatial conjunction of formulas of quantifier depth one preserves the decidability of two-variable logic with counting. This result applies to two-variable role logic fragment as well.
The resulting logic smoothly integrates type system and predicate calculus notation and can be viewed as a natural generalization of the notation for constraints arising in role analysis and similar shape analysis approaches.

Keywords:. Records, Shape Analysis, Static Analysis, Program Verification, Two-Variable Logic with Counting, Description Logic, Types

Table of Contents

Version of April 6, 2004, 1:38pm.

1 Introduction

In [36] we have introduced *role logic*, a notation for describing properties of relational structures in shape analysis, databases and knowledge bases. Role logic notation aims to combine the simplicity of role declarations [33] and the well-established first-order logic. Role logic is closed under all boolean operations and generalizes boolean shape analysis constraints [37]. Role logic formulas easily translate into the traditional first-order logic notation. Despite this generality, role logic enables the concise expression of common properties of data structures in imperative programs that manipulate complex data structures with mutable references. In [36, Section 4] we have established the decidability of the fragment RL^2 of role logic by exhibiting a correspondence with two-variable logic with counting C^2 [22, 45].

Generalized records in role logic. In this paper we give a systematic account of field and slot declarations of role analysis [33] by introducing a set of role logic shorthands that allows concise description of records. Our basic idea is to generalize types to unary predicates on objects. Some of the aspects of our notion of records that indicate its generality are:

1. We allow building new records by taking the conjunction, disjunction, or negation of records.
2. In our notation, a record indicates a property of an object at a particular program point; objects can satisfy different record specifications at different program points. As a result, our records can express typestate changes such as object initialization [16–18, 55, 56] and more general changes in relationships between objects such as movements of objects between data structures [32, 33, 54].
3. We allow *inverse records* as a dual of records that specify incoming edges of an object in the graph of objects representing program heap. Inverse records allow the specification of aliasing properties of objects, generalizing unique pointers. Inverse records enable the convenient specification of movements of objects that participate in multiple data structures.
4. We allow the specification of both open and closed records. Closed records specify a complete set of outgoing and incoming edges of an object. Open records leave certain edges unspecified, which allows orthogonal data structures to be specified independently and then combined using logical conjunction.
5. We allow the concatenation of generalized records using a form of spatial conjunction of separation logic, while remaining within the decidable fragment of two-variable role logic.

Separation logic. Separation logic [28, 43, 51, 52] is a promising approach for specifying properties of programs in the presence of mutable data structures. One of the main uses of separation logic in previous approaches is dealing with frame conditions [5, 28]. In contrast, our paper identifies another use of spatial logic: expressing record concatenation. Although our approach is based on essentially same logical operation of spatial conjunction, our use of spatial conjunction for records is more local, because it applies to the descriptions of the neighborhood of an object.

To remain within the decidable fragment of role logic, we give in Section 7 a construction that eliminates spatial conjunction when it connects formulas of quantifier depth one. This construction also illustrates that spatial conjunction is useful for reasoning about counting stars [22] of the two-variable logic with counting C^2. To our knowledge, this is the first result that combines two-variable logic with counting and a form of spatial conjunction.

Using the resulting logic. We can use specifications written in our notation to describe properties and relations between objects in programs with dynamically allocated data structures. These specifications can act as assertions, preconditions, postconditions, loop invariants or data structure invariants [33, 36, 39]. By selecting a finite-height lattice of properties for a given program fragment, abstract interpretation [15] can be used to synthesize properties of objects at intermediate program points [2, 3, 24, 33, 49, 50, 54, 58, 59]. Decidability and closure properties of our notation are essential for the completeness and predictability of the resulting static analysis [38].

Contributions. We summarize the main contributions of this paper as follows:

1. We present a logic which generalizes the concept of records in several directions (Section 5). These generalizations are useful for expressing properties of objects and memory cells in imperative programs, and go beyond standard type systems.
2. We identify a novel use of separation logic: modelling the concatenation of generalized records.
3. We show how to translate role constraints from role analysis [33] to role logic (Section 6).
4. We show that, under certain syntactic restrictions, we can translate spatial conjunction into other constructs of the decidable logic RL^2 (Section 7). We therefore obtain a notation that extends RL^2 with a convenient way of describing record concatenation, and remains decidable.
5. We present a translation of first-order logic with spatial conjunction and inductive definitions into second-order logic (Section 8.2).

Outline. Section 2 reviews the syntax and semantics of role logic. Section 3 defines spatial conjunction in role logic and motivates its use for describing record concatenation. Section 4 and Section 5 show how to use spatial conjunction in role logic to describe a generalization of records. Section 6 demonstrates that our notation is a generalization of the local constraints arising in role analysis [33] by giving a natural embedding of role constraints into our notation. Section 7 shows how to eliminate the spatial conjunction connective $\circledast$ from a spatial conjunction $F_1 \circledast F_2$ of two formulas F_1 and F_2 when F_1 and F_2 have no nested counting quantifiers; this is the core technical result of this paper. A consequence of this is result is that we may allow certain uses of spatial conjunction in RL^2 fragment of role logic while preserving the decidability property of RL^2. Our extension of role logic with spatial conjunction is therefore justified: it allows record-like specifications to be expressed in a more natural way, and it does not lead outside the decidable fragment. Section 8 contains remarks on preserving the satisfiability of formulas in the presence of spatial conjunction and shows how to encode the spatial conjunction (with inductive definitions) in second-order logic. Section 9 presents related work, and Section 10 concludes. Appendix contains the details of the correctness proof for the elimination of spatial conjunction from Section 7.

2 A Decidable Two-Variable Role Logic RL^2

$$\begin{array}{rcll}
F & ::= & A \mid f \mid \mathsf{EQ} \mid F_1 \wedge F_2 \mid \neg F \mid F' \mid {\sim}F \mid \mathsf{card}^{\geq k} F & \\
e & :: & \{1,2\} \rightarrow D & \\
\llbracket A \rrbracket e & = & \llbracket A \rrbracket (e\,1) & \llbracket f \rrbracket e = \llbracket f \rrbracket (e\,2, e\,1) \\
\llbracket \mathsf{EQ} \rrbracket e & = & (e\,2) = (e\,1) & \\
\llbracket F_1 \wedge F_2 \rrbracket e & = & (\llbracket F_1 \rrbracket e) \wedge (\llbracket F_2 \rrbracket e) & \llbracket \neg F \rrbracket e = \neg(\llbracket F \rrbracket e) \\
\llbracket F' \rrbracket e & = & \llbracket F \rrbracket (e[1 \mapsto (e\,2)]) & \llbracket {\sim}F \rrbracket e = \llbracket F \rrbracket (e[1 \mapsto (e\,2), 2 \mapsto (e\,1)]) \\
\llbracket \mathsf{card}^{\geq k} F \rrbracket e & = & |\{d \in D \mid \llbracket F \rrbracket (e[1 \mapsto o, 2 \mapsto (e\,1)])\}| \geq k &
\end{array}$$

Fig. 1. The Syntax and the Semantics of RL^2

Figure 1 presents the two-variable role logic RL^2 [36]. We have proved in [36] that RL^2 has the same expressive power as two-variable logic with counting C^2. The logic C^2 is a first-order logic 1) extended with counting quantifiers $\exists^{\geq k}x.F(x)$, saying that there are at least k elements x satisfying formula $F(x)$ for some constant k, and 2) restricted to allow only two variable names x, y in formulas. An example formula in two-variable logic with counting is

$$\forall x.A(x) \Rightarrow (\forall y.f(x,y) \Rightarrow \exists^{=1}x.\, g(x,y)) \tag{1}$$

The formula (1) means that all nodes that satisfy $A(x)$ point along the field f to nodes that have exactly one incoming g edge. Note that the variables x and y may be reused via quantifier nesting, and that formulas of the form $\exists^{=k}x.\, F(x)$ and $\exists^{\leq k}x.\, F(x)$ are expressible as boolean combination of formulas of the form $\exists^{\geq k}x.\, F(x)$. The logic C^2 was shown decidable in [22] and the complexity for the C^2_1 fragment of C^2 (with counting up to one) was established in [45]. We can view role logic as a variable-free version of C^2. Variable-free logical notations are attractive as generalizations of type systems because traditional type systems are often variable-free. The formula (1) can be written in role logic as $[A \Rightarrow [f \Rightarrow \mathsf{card}^{\geq 1}{\sim}g]]$ where the construct $[F]$ is a shorthand for $\neg\mathsf{card}^{\geq 1}\neg F$ and corresponds to the universal quantifier. The expression ${\sim}g$ denotes the inverse of relation g. This paper focuses on the use of role logic to describe generalized records, see [36] for further examples of using role logic and [6] for advantages of variable-free notation in general.

3 Spatial Conjunction

$$\llbracket F_1 \circledast F_2 \rrbracket e = \exists e_1, e_2.\ \mathsf{split}\, e\, [e_1\, e_2] \ \wedge\ \llbracket F_1 \rrbracket e_1 \wedge \llbracket F_2 \rrbracket e_2$$

$$\mathsf{split}\, e\, [e_1\, e_2] =$$

$$\quad \forall A \in \mathcal{A}.\ \forall d \in D.\ (e\, A)\, d \iff (e_1\, A)\, d \vee (e_2\, A)\, d \quad \wedge\ \neg((e_1\, A)\, d \wedge (e_2\, A)\, d) \ \wedge$$

$$\quad \forall f \in \mathcal{F}.\ \forall d_1, d_2 \in D.$$

$$\qquad (e\, f)\, d_1\, d_2 \iff (e_1\, f)\, d_1\, d_2 \vee (e_2\, f)\, d_1\, d_2 \quad \wedge\ \neg((e_1\, f)\, d_1\, d_2 \wedge (e_2\, f)\, d_1\, d_2)$$

$$\mathsf{emp} \ \equiv\ [[\bigwedge_{A \in \mathcal{A}} \neg A \ \wedge \ \bigwedge_{f \in \mathcal{F}} \neg f]]$$

priority: $\wedge$ binds strongest, then $\circledast$, then $\vee$

$$F \ \sim\ G \ \text{ means } \ \forall e. \llbracket F \rrbracket e = \llbracket G \rrbracket e$$

$$(F_1 \circledast F_2) \circledast F_3 \ \sim\ F_1 \circledast (F_2 \circledast F_3)$$

$$F \circledast \mathsf{emp} \ \sim\ \mathsf{emp} \circledast F \ \sim\ F$$

$$F_1 \circledast F_2 \ \sim\ F_2 \circledast F_1$$

$$F_1 \circledast (F_2 \vee F_3) \ \sim\ F_1 \circledast F_2 \ \vee\ F_1 \circledast F_3$$

Fig. 2. Semantics and Properties of Spatial Conjunction $\circledast$.

Figure 2 shows our semantics of spatial conjunction $\circledast$. To motivate our use of spatial conjunction, we first illustrate how role logic supports the description of simple properties of objects in a concise way. Indeed, one of the design goals of role logic is to have a logic-based specification language where simple properties of objects are as convenient to write as type declarations in a language like Java.

Example 1. The formula $[f \Rightarrow A]$ is true for an object whose every f-fields points to an A object, $[g \Rightarrow B]$ means that every g-field points to a B object, so

$$[f \Rightarrow A] \wedge [g \Rightarrow B]$$

denotes the objects that has both f pointing to an A object and g pointing to a B object. Such specification is as concise as the following Java class declaration

```
class C { A f; B g; }
```

Example 1 illustrates how the presence of conjunction $\wedge$ in role logic enables combination of orthogonal properties such as constraints on distinct fields. However, not all properties naturally compose using conjunction.

Example 2. Consider a program that contains three fields, modelled as binary relations f, g, h. The formula $P_f \;\equiv\; (\mathsf{card}^{=1} f) \land (\mathsf{card}^{=0}(g \lor h))$ means that the object has only one outgoing f-edge and no other edges. The formula $P_g \;\equiv\; (\mathsf{card}^{=1} g) \land (\mathsf{card}^{=0}(f \lor h))$ means that the object has only one outgoing g-edge and no other edges. If we "physically join" two records, each of which has one field, we obtain a record that has two fields, and is described by the formula

$$P_{fg} \;\equiv\; (\mathsf{card}^{=1} f) \land (\mathsf{card}^{=1} g) \land (\mathsf{card}^{=0} h)$$

Note that it is *not* the case that $P_{fg} \;\sim\; P_f \land P_g$. More generally, no boolean combination of P_f and P_g yields P_{fg}.

Example 2 prompts the question: is there an operation that allows joining specifications that will allow us to combine P_f and P_g into P_{fg}? Moreover, can we define such an operation on records viewed as arbitrary formulas in role logic?

It turns out that there is a natural way to describe the set of models of formula P_{fg} in Example 2 as the result of "physically merging" the edges (relations) of the models of P_f and models of P_g. The merging of disjoint models of formulas is the idea behind the definition of spatial conjunction $\circledast$ in Figure 2. The predicate $(\mathsf{split}\, e\, [e_1\, e_2])$ is true iff the relations of the model (environment) e can be split into e_1 and e_2 and the notation generalizes to splitting into any number of environments.

Example 3. For P_f, P_g, and P_{fg} of Example 2, we have $P_{fg} = P_f \circledast P_g$.

Note that the operation $\circledast$ is associative and commutative. The formula emp, which asserts that all predicates are false, is the unit for $\circledast$. Moreover, $\circledast$ distributes over $\lor$.

A note on relationship with [28]. The semantics of spatial conjunction in Figure 2 match the semantics of [28], with two differences.

A small technical difference is that Figure 2 splits the edges of the model (the tuples of the relations), whereas [28] splits the domain. The difference arises because the elements of the domain in [28] are locations, whereas the elements of our models are objects. To represent a location in our view, we would use a tuple $\langle o, f\rangle$ where o is an element of the domain and f is a field name.

A higher-level difference is that the use of spatial logic we propose in this paper is the notation for records (Section 5), as opposed to the description of global heap properties. When used for formulas of quantifier depth one (Section 7), spatial conjunction does not even change the set of definable relations of two-variable logic with counting.

4 Field Complement

As a step towards record calculus in role logic, this section introduces the notion of a *field complement*, which makes it easier to describe records in role logic.

Example 4. Consider the formula $P_f \ \equiv\ (\mathsf{card}^{=1} f) \wedge (\mathsf{card}^{=0}(g \vee h))$ from Example 2, stating the property that an object has only one outgoing f-edge and *no other edges.* Property P_f has little to do with g or h, yet g and h explicitly occur in P_f. Moreover, we need to know the entire set of relations in the language to write P_f; if the language contains an additional field i, the property P_f would become $P_f \ \equiv\ (\mathsf{card}^{=1} f) \wedge (\mathsf{card}^{=0}(g \vee h \vee i))$. Note also that $\neg f$ is not the same as $g \vee h \vee i$, because $\neg f$ computes the complement of the value of the relation f with respect to the universal set, whereas $g \vee h \vee i$ is the union of all relations other than f.

To address the notational problem illustrated in Example 4, we introduce the symbol edges, which denotes the union of all binary relations, and the notation $-f$ (*field complement* of f), which denotes the union of all relations other than f.

$$\mathsf{edges} \equiv \bigvee_{g} g \qquad\qquad -f \equiv \bigvee_{g \neq f} g$$

This additional notation allows us to avoid explicitly listing all fields in the language when stating properties like P_f.

Example 5. Formula P_f from Example 4 can be written as $P_f \ \equiv\ (\mathsf{card}^{=1} f) \wedge (\mathsf{card}^{=0} {-f})$, which mentions only f. Even when the language is extended with additional relations, P_f still denotes the intended property. Similarly, to denote the property of an object that has outgoing fields given by P_f and has no incoming fields, we use the predicate $P_f \wedge \mathsf{card}^{=0} {\sim}\mathsf{edges}$.

We use the notation edges and $-f$ to build the notation for records and inverse records in Section 5 below.

A note on ternary relation interpretation. It is possible to provide a notation for relations that generalizes the notation edges and $-f$. The idea of this generalization is to change the definition of the model (environment). Instead of a model that specifies a binary relation for each field, the model specifies the value of one ternary relation H and a unary tag-predicate for each field name. For example, instead of the model that provides interpretations f_I and g_I for two binary relations f and g, we could use the model that provides interpretation of $[\![H]\!]$, where

$$\begin{aligned} [\![H]\!] o_1\, o_2\, n \ &= \ (n{=}f_0 \wedge f_I\, o_1\, o_2) \ \vee \\ & \quad\ (n{=}g_0 \wedge f_I\, o_1\, o_2) \end{aligned}$$

and the interpretation of unary tag-predicates f and g. Here f_0 is an element of the domain that tags tuples coming from $[\![f]\!]$, whereas g_0 tags tuples coming from $[\![g]\!]$. We interpret f as a predicate that is true only on the element f_0, and similarly g as a predicate true only on the element g_0. We then introduce the following dereferencing shorthand:

$$\uparrow F \ \equiv\ \{H \wedge F\} \tag{2}$$

The expression $\uparrow f$ now denotes the original interpretation of f, that is, $[\![\uparrow f]\!] = f_I$. Moreover, $\uparrow\neg f$ corresponds to field complement $-f$, and $\uparrow\mathsf{True}$ corresponds to edges. Note that the expressions of the form $\uparrow(\neg f \wedge \neg g)$ are now also available. Let B be a boolean combination of unary predicates denoting fields. These unary predicates are disjoint, so transforming B into disjunctive normal form and applying the property

$$\uparrow(B_1 \vee B_2) = \uparrow B_1 \vee \uparrow B_2$$

which follows from (2), allows transforming $\uparrow B$ into a boolean combination of expressions of the form $\uparrow f$ and $\uparrow g$. This means that we obtain no additional expressive power using expressions of the form $\uparrow B$ where B is a boolean combination of unary predicates denoting fields, so for simplicity we do not consider such "ternary relation interpretation" further in this paper.

5 Records and Inverse Records

In this section we use role logic with spatial conjunction and field complement from Section 4 to introduce a notation for records. We also introduce inverse records, which are dual to records, and correspond to slot constraints in role analysis [33].

$$\begin{aligned}
\text{multifield: } & f \xrightarrow{*} A \equiv \mathsf{card}^{=0}(-f \vee (f \wedge \neg A)) \\
\text{field: } & f \xrightarrow{s} A \equiv \mathsf{card}^{s}(A \wedge f) \ \wedge \ f \xrightarrow{*} A \\
& \qquad s \text{ of the form } {=}k, {\leq}k, \text{ or } {\geq}k, \text{for } k \in \{0,1,2,\ldots\} \\
& f \rightarrow A \equiv f \xrightarrow{=1} A \\
\text{multislot: } & A \xleftarrow{*} f \equiv \mathsf{card}^{=0}(\sim{-f} \vee (\sim f \wedge \neg A)) \\
\text{slot: } & A \xleftarrow{s} f \equiv \mathsf{card}^{s}(A \wedge \sim f) \ \wedge \ A \xleftarrow{*} f \\
& \qquad s \text{ of the form } {=}k, {\leq}k, \text{ or } {\geq}k, \text{for } k \in \{0,1,2,\ldots\} \\
& A \leftarrow f \equiv A \xleftarrow{=1} f
\end{aligned}$$

$$\begin{aligned}
\mathsf{fm} &::= \mathsf{field} \mid \mathsf{multifield} \\
\mathsf{closedRecord} &::= \mathsf{fm} \mid \mathsf{closedRecord} \circledast \mathsf{fm} \\
\mathsf{openRecord} &::= \mathsf{closedRecord} \circledast \mathsf{True}
\end{aligned}$$

$$\begin{aligned}
\mathsf{sm} &::= \mathsf{slot} \mid \mathsf{multislot} \\
\mathsf{closedInvRecord} &::= \mathsf{sm} \mid \mathsf{closedInvRecord} \circledast \mathsf{sm} \\
\mathsf{openInvRecord} &::= \mathsf{closedInvRecord} \circledast \mathsf{True}
\end{aligned}$$

Fig. 3. Record Notation

Figure 3 presents the notation for records and inverse records. A *field* predicate $f \rightarrow A$ is true for an object whose only outgoing edge in the graph (model) is an f-edge terminating at A. Dually, a *slot* predicate $A \leftarrow f$ is true for an object whose only incoming edge in the graph is an f-edge originating at A. A *multifield* predicate $f \xrightarrow{*} A$ is true iff the object has any number of outgoing f-edges terminating at A, and no other edges. Dually, a *multislot* predicate $A \xleftarrow{*} f$ is true iff the object has any number of incoming f-edges originating from A, and no other edges. We also allow notation $f \xrightarrow{s} A$ where s is an expression of the form $=k$, $\leq k$, or $\geq k$. This notation gives a bound on the number of outgoing edges, and implies that there are no other outgoing edges. We similarly introduce $A \xleftarrow{s} f$. A closed record is a spatial conjunction of fields and multifields. An open record is a spatial conjunction of a closed record with True. While a closed record allows only the listed fields, an open record allows any number of additional fields. Inverse records are dual to records, and we similarly distinguish open and closed inverse records.

Example 6. To describe a closed record whose only fields are f and g where f-fields point to objects in the set A and g-fields point to objects in the set B, we use the predicate $P_1 \equiv f \rightarrow A \circledast g \rightarrow B$. The definition of P_1 lists all fields of the object. To specify an open record which certainly has fields f and g but may or may not have other fields, we write $P_2 \equiv f \rightarrow A \circledast g \rightarrow B \circledast \mathsf{True}$. Neither P_1 nor P_2 restrict incoming references of an object. To specify that the only incoming references of an object are from the field h, we conjoin P_2 with the closed inverse record consisting of a single multislot $\mathsf{True} \xleftarrow{*} h$, yielding the predicate $P_3 \equiv P_2 \land \mathsf{True} \xleftarrow{*} h$. To specify that an object has exactly one incoming reference, and that the incoming reference is from the h field and originates from an object belonging to the set C, we use $P_4 \equiv P_2 \land C \leftarrow h$. Note that specifications P_3 and P_4 go beyond most standard type systems in their ability to specify the incoming (in addition to the outgoing) references of objects.

6 Role Constraints

Role constraints were introduced in [30, 31, 33]. In this section we show that role logic is a natural generalization of role constraints by giving a translation from role constraints to role logic. A logical view of role constraints is also suggested in [35, 35]. A role is a set of objects that satisfy a conjunction of the following four kinds of constraints: field constraints, slot constraints, identities, acyclicities. In this paper we show that role logic naturally models field constraints, slot constraints, and identities. [1]

Roles describing complete sets of fields and slots. Figure 4 shows the translation of role constraints [33, Section 3] into role logic formulas. The simplicity of the translation is a consequence of the notation for records that we have developed in this paper.

[1] Acyclicities go beyond first-order logic because they involve non-local transitive closure properties.

$$
\begin{aligned}
\mathcal{C}[\![\texttt{fields } F;\ \texttt{slots } S;\ \texttt{identities } I;\ \texttt{acyclic } A]\!] &= \mathcal{C}[\![\texttt{fields } F]\!] \ \wedge\ \mathcal{C}[\![\texttt{slots } S]\!] \ \wedge \\
&\qquad [\![\texttt{identities } I]\!] \ \wedge\ [\![\texttt{acyclic } A]\!]
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{C}[\![\texttt{fields } f_1 : S_1, \ldots, f_n : S_n]\!] &= f_1 \rightarrow S_1 \ \circledast\ \ldots\ \circledast\ f_n \rightarrow S_n \\
\mathcal{C}[\![\texttt{slots } S_1.f_1, \ldots, S_n.f_n]\!] &= S_1 \leftarrow f_1 \ \circledast\ \ldots\ \circledast\ S_n \leftarrow f_n \\
[\![\texttt{identities } f_1.g_1, \ldots, f_n.g_n]\!] &= \textstyle\bigwedge_{i=1}^{n} [f_i \Rightarrow {\sim} g_i] \\
[\![\texttt{acyclic } f_1, \ldots, f_n]\!] &= \texttt{acyclic}\ \ (\textstyle\bigvee_{i=1}^{n} f_i)
\end{aligned}
$$

Fig. 4. Translation of Role Constraints [33] into Role Logic Formulas

$$
\begin{aligned}
\mathcal{O}[\![\texttt{fields } F;\ \texttt{slots } S;\ \texttt{identities } I;\ \texttt{acyclic } A]\!] &= \mathcal{O}[\![\texttt{fields } F]\!] \ \wedge\ \mathcal{O}[\![\texttt{slots } S]\!] \ \wedge \\
&\qquad [\![\texttt{identities } I]\!] \ \wedge\ [\![\texttt{acyclic } A]\!] \\
\mathcal{O}[\![\texttt{fields } f_1 : S_1, \ldots, f_n : S_n]\!] &= \mathcal{C}[\![\texttt{fields } f_1 : S_1, \ldots, f_n : S_n]\!] \circledast \mathsf{card}^{=0}(\textstyle\bigvee_{i=1}^{n} f_i) \\
\mathcal{O}[\![g_1, \ldots, g_m \texttt{ slots } S_1.f_1, \ldots, S_n.f_n]\!] &= \mathcal{C}[\![\texttt{slots } S_1.f_1, \ldots, S_n.f_n]\!] \circledast \mathsf{card}^{=0}(\textstyle\bigvee_{i=1}^{m} {\sim} g_i)
\end{aligned}
$$

Fig. 5. Translation of Simultaneous Role Constraints [33, Section 7.2] into Role Logic Formulas. See also Figure 4.

Simultaneous Roles. In object-oriented programs, objects may participate in multiple data structures. The idea of simultaneous roles [33, Section 7.2] is to associate one role for the participation of an object in one data structure. When the object participates in multiple data structures, the object plays multiple roles. Role logic naturally models simultaneous roles: each role is a unary predicate, and if an object satisfies multiple roles, the the object satisfies the conjunction of predicates. Figure 5 presents the translation of field and slot constraints of simultaneous roles into role logic. Whereas the roles of [33, Section 3] translate to closed records and closed inverse records, the simultaneous roles of [33, Section 7.2] translate specifications that are closer to open records and open inverse records.

7 Eliminating Spatial Conjunction in RL^2

Preserving the decidability. Previous sections have demonstrated the usefulness of adding record concatenation in the form of spatial conjunction to our notation for generalized records. However, a key question remains: is the resulting extended notation decidable? In this section we give an affirmative answer to this question by showing how to compute the spatial conjunction using the remaining logical operations for a large class of record specifications.

Approach. Consider two formulas F_1 and F_2 in first-order logic with counting, where both F_1 and F_2 have quantifier depth one. An equivalent way of stating the condition on F_1 and F_2 is that there are no nested occurrences of quantifiers. (Note that we count one application of $\exists^{\geq k}x. P$ as one quantifier, regardless of the value k.) We show that, under these conditions, the spatial conjunction $F_1 \circledast F_2$ can be written as an equivalent formula F_3 where F_3 does not contain the spatial conjunction operation $\circledast$. The proof proceeds by writing formulas F_1, F_2 in a normal form, as a disjunction of counting stars [22], and showing that the spatial conjunction of counting stars is equivalent to a disjunction of counting stars.

As a consequence of the results in this section, adding the operation $\circledast$ to logic with counting does not change its expressive power provided that both F_1 and F_2 have quantifier depth at most one. Here we allow F_1 and F_2 themselves to contain spatial conjunction, because we may eliminate spatial conjunction in F_1 and F_2 recursively. Applying these results to two-variable logic with counting C^2, we conclude that introducing into C^2 the spatial conjunction of formulas of quantifier depth one preserves the decidability of C^2. Furthermore, thanks to the translations between C^2 and RL^2 in [36], if we allow the spatial conjunction of RL^2 formulas with no nested card occurrences, we preserve the decidability of the logic RL^2. The formulas of the resulting logic are given by

$$\begin{aligned} F ::= {} & A \mid f \mid \mathsf{EQ} \mid F_1 \wedge F_2 \mid \neg F \mid F' \mid {\sim}F \mid \mathsf{card}^{\geq k} F \\ \mid {} & F_1 \circledast F_2, \text{ if } F_1 \text{ and } F_2 \text{ have no nested } \mathsf{card} \text{ occurrences} \end{aligned}$$

Note that record specifications in Figure 3 contain no nested card occurrences, so joining them using $\circledast$ yields formulas in the decidable fragment. Hence, in addition to quantifiers and boolean operations, the resulting logic supports a generalization of record concatenation, and is still decidable; this decidability property is what we show in the sequel. We present the sketch of the proof, see Appendix for proof details..

7.1 Atomic Type Formulas

In this section we introduce classes of formulas that correspond to the model-theoretic notion of atomic type [44, Page 20] (see [25, Page 42] and [12, Page 78] for the notion of type in general). We then introduce formulas that describe the notion of counting stars [22, 45]. We conclude this section with Proposition 12, which gives the normal form for formulas of quantifier depth one.

If $\mathcal{C} = C_1, \ldots, C_m$ is a finite set of formulas, then a *cube over* $\mathcal{C}$ is a conjunction of the form $C_1^{\alpha_1} \wedge \ldots C_m^{\alpha_m}$ where $\alpha_i \in \{0, 1\}$, $C^1 = C$ and $C^0 = \neg C$. For simplicity, fix a finite language $L = \mathcal{A} \cup \mathcal{F}$ with $\mathcal{A}$ a finite set of unary predicate symbols and $\mathcal{F}$ a finite set of binary predicate symbols. We work in predicate calculus with equality, and assume that the equality "=", where $= \notin \mathcal{F}$, is present as a binary relation symbol, unless explicitly stated otherwise. We use D to denote a finite domain of interpretation and e to denote a model with variable assignment; e maps $\mathcal{A}$ to 2^D, maps $\mathcal{F}$ to $2^{D \times D}$ and maps variables to elements of D. Let $x_1, \ldots, x_n$ be a finite list of distinct variables. Let $\mathcal{C}$ be the set of all atomic formulas F such that $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$. The set $\mathcal{C}$ is finite (in our case it has $|\mathcal{A}|n + (|\mathcal{F}|+1)n^2$ elements). We call a cube over $\mathcal{C}$ a *complete atomic type (CAT) formula*.

Example 7. If $\mathcal{A} = \{A\}$ and $\mathcal{F} = \{f\}$, then

$$A(x_1) \wedge \neg A(x_2) \wedge$$
$$\neg f(x_1, x_1) \wedge \neg f(x_2, x_2) \wedge f(x_1, x_2) \wedge \neg f(x_2, x_1) \wedge$$
$$x_1 = x_1 \wedge x_2 = x_2 \wedge x_1 \neq x_2 \wedge x_2 \neq x_1$$

is a CAT formula.

We may treat conjunction of literals as the set of literals, so we say that "a literal belongs to the conjunction" and apply set-theoretic operations on conjunctions of literals.

From the disjunctive normal form theorem for propositional logic, we obtain the following Proposition 8.

Proposition 8. *Every quantifier-free formula F such that $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$ is equivalent to a disjunction of CAT formulas C such that $\mathsf{FV}(C) = \{x_1, \ldots, x_n\}$.*

A CAT formula may be contradictory if, for example, it contains the literal $x_i \neq x_i$ as a conjunct. We next define classes of CAT formulas that are satisfiable in the presence of equality. Let $x_1, \ldots, x_n$ be distinct variables. A *general-case CAT (GCCAT)* formula is a CAT formula F such that the following two conditions hold: 1) $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$; 2) for all $1 \leq i, j \leq n$, the conjunct $x_i = x_j$ is in F iff $i \equiv j$. Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ be distinct variables. An *equality CAT (EQCAT)* formula is a formula of the form $\bigwedge_{j=1}^{m} y_j = x_{i_j} \;\wedge\; F$, where $1 \leq i_1, \ldots, i_m \leq n$ and F is a GCCAT formula such that $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$.

Lemma 9. *Every CAT formula F is either contradictory, or is equivalent to an EQCAT formula F' such that $\mathsf{FV}(F') = \mathsf{FV}(F)$.*

From Proposition 8 and Lemma 9, we obtain the following Proposition 10.

Proposition 10. *Every quantifier-free formula F such that $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$ can be written as a disjunction of EQCAT formulas C such that $\mathsf{FV}(C) = \{x_1, \ldots, x_n\}$.*

We next introduce the notion of an extension of a GCCAT formula. Let $x, x_1, \ldots, x_n$ be distinct variables and F be a GCCAT formula such that $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$. We say that F' is an x-extension of F, and write $F' \in \mathsf{exts}(F, x)$ iff all of the following conditions hold: 1) $F \wedge F'$ is a GCCAT formula; 2) $\mathsf{FV}(F \wedge F') = \{x, x_1, \ldots, x_n\}$; 3) F and F' have no common atomic formulas. Note that if $\mathsf{FV}(F_1) = \mathsf{FV}(F_2)$, then $\mathsf{exts}(F_1, x) = \mathsf{exts}(F_2, x)$ i.e. the set of extensions of a GCCAT formula depends only on the free variables of the formula; we introduce additional notation $\mathsf{exts}(x_1, \ldots, x_n, x)$ to denote $\mathsf{exts}(F, x)$ for $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$.

To define a normal form for formulas of quantifier depth one, we introduce the notion of k-counting star. If $p \geq 2$ is a non-negative integer, let p^+ be a new symbol which represents the co-finite set of integers $\{p, p+1, \ldots\}$. Let $C_p = \{0, 1, \ldots, p{-}1, p^+\}$. If $c \in C_p$, by $\exists^i x.\, P$ we mean $\exists^{=i} x.\, P$ if i is an integer, and $\exists^{\geq p} x.\, P$ if $i = p^+$. We say that a formula F has a *counting degree* of at most p iff the only counting quantifiers in F are of the form $\exists^c x.\, G$ for some $c \in C_{p+1}$.

Definition 11 (Counting Star Formula). *Let x, $x_1, \ldots, x_n$, and $y_1, \ldots, y_m$ be distinct variables, $k \geq 1$ a positive integer, and F a GCCAT formula such that $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$. A* k-counting star function for F *is a function $\gamma : \mathsf{exts}(F, x) \to C_{k+1}$. A* k-counting-star formula for γ *is a formula of the form*

$$\bigwedge_{j=1}^{m} y_j = x_{i_j} \ \wedge \ F \ \wedge \bigwedge_{F' \in \mathsf{exts}(F,x)} \exists^{\gamma(F')} x.\, F'$$

where $1 \leq i_1, \ldots, i_m \leq n$.

Note that in Definition 11, formula $\bigwedge_{j=1}^{m} y_j = x_{i_j} \wedge F$ is an EQCAT formula, and formula $\bigwedge_{j=1}^{m} y_j = x_{i_j} \wedge F \wedge F'$ is an EQCAT formula for each $F' \in \mathsf{exts}(F, x)$.

The following Proposition 12 shows that formulas of quantifier depth at most one are equivalent to disjunctions of counting stars.

Proposition 12 (Depth-One Normal Form). *Let F be a formula of such that F has quantifier depth at most one, F has counting degree at most k, and $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$. Then F is equivalent to a disjunction of k-counting-star formulas F_C where $\mathsf{FV}(F_C) = \{x_1, \ldots, x_n\}$.*

7.2 Spatial Conjunction of Stars

Sketch of the construction. Let F_1 and F_2 be two formulas of quantifier depth at most one, and not containing the logical operation $\circledast$. By Proposition 12, let F_1 be equivalent to the disjunction of counting star formulas $\bigvee_{i=1}^{n_1} C_{1,i}$ and let F_2 be equivalent to the disjunction of counting star formulas $\bigvee_{j=1}^{n_2} C_{2,j}$. By distributivity of law of $\circledast$ with respect to $\vee$, we have

$$F_1 \circledast F_2 \ \sim \ (\bigvee_{i=1}^{n_1} C_{1,i}) \circledast (\bigvee_{j=1}^{n_2} C_{2,j}) \ \sim \ \bigvee_{i=1}^{n_1} \bigvee_{j=1}^{n_2} C_{1,i} \circledast C_{2,j}$$

In the sequel we show that a spatial conjunction of counting-star formulas is either contradictory or is equivalent to a disjunction of counting star formulas. This suffices to eliminate spatial conjunction of formulas of quantifier depth at most one. Moreover, if F is any formula of quantifier depth at most one, possibly containing $\circledast$, by repeated elimination of the innermost $\circledast$ we obtain a formula without $\circledast$.

To compute the spatial conjunction of counting stars we establish an alternative syntactic form for counting star formulas. The idea of this alternative form is roughly to replace a counting quantifier such as $\exists^{=k} x.\, F'$ with a spatial conjunction of k formulas each of which has the meaning similar to $\exists^{=1} x.\, F'$, and then combine a formula $\exists^{=1}x.\, F_1'$ resulting from one counting star with a formula $\exists^{=1}x.\, F_2'$ resulting from another counting star into the formula $\exists^{=1}x.\, (F_1' \odot F_2')$ where $\odot$ denotes merging of GCCAT formulas by taking the union of their positive literals. We next develop this idea in greater detail.

Notation for spatial representation of stars. Let $G_E(x_1, \ldots, x_n)$ be the unique GCCAT formula F with $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$ such that the only positive literals in F are literals $x_i = x_i$ for $1 \leq i \leq n$. Similarly, there is a unique formula $F' \in \mathsf{exts}(x_1, \ldots, x_n, x)$ such that every atomic formula in F' distinct from for $x = x$ occurs in a negated literal. We call F' an *empty extension* and denote it $\mathsf{empEx}(x_1, \ldots, x_n, x)$.

To compute a spatial conjunction of formulas C_1 and C_2 in the language L, we temporarily consider formulas in an extended language $L' = L \cup \{B_1, B_2\}$ where B_1 and B_2 are two new unary predicates used to mark formulas. We use B_1 to mark formulas derived from C_1, and use B_2 to mark formulas derived from C_2. For $m \in \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$, define

$$\begin{array}{ll} \mathsf{Mark}_\emptyset(x) = \neg B_1(x) \wedge \neg B_2(x) & \mathsf{Mark}_1(x) = B_1(x) \wedge \neg B_2(x) \\ \mathsf{Mark}_2(x) = \neg B_1(x) \wedge B_2(x) & \mathsf{Mark}_{1,2}(x) = B_1(x) \wedge B_2(x) \end{array}$$

Note that, when we say that F is a GCCAT formula, we mean that F is GCCAT formula in language L (and thus F mentions symbols only from L), even when we use F as a subformula of a larger formula in language L'. Similarly, expressions $\mathsf{exts}(x_1, \ldots, x_n, x)$, $\mathsf{empEx}(F, x)$, and $G_E(x_1, \ldots, x_n)$ all denote formulas in language L.

On the other hand, $\mathsf{empEx}_\emptyset(F, x)$ and empe are formulas in language L'. Formula $\mathsf{empEx}_\emptyset(F, x)$ is an empty extension of F in language L'. Formula empe asserts that $x_1, \ldots, x_n$ have an empty GCCAT formula and that the remaining elements have empty extension in L'. Formula empe does not constrain the values $B_1(x_i)$ and $B_2(x_i)$, these values turn out to be irrelevant.
Let $F' \in \mathsf{exts}(x_1, \ldots, x_n, x)$. Define

$$\mathsf{empEx}_\emptyset(x_1, \ldots, x_n, x) \equiv \mathsf{empEx}(x_1, \ldots, x_n, x) \wedge \mathsf{Mark}_\emptyset(x)$$

$$\mathsf{empe}(x_1, \ldots, x_n) \equiv G_E(x_1, \ldots, x_n) \ \wedge\ \forall x.\, (\textstyle\bigwedge_{i=1}^n x \neq x_i) \Rightarrow \mathsf{empEx}_\emptyset(x_1, \ldots, x_n, x)$$

We write $\mathsf{empEx}_\emptyset(F, x)$ for $\mathsf{empEx}_\emptyset(x_1, \ldots, x_n, x)$ if $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$, and similarly for $\mathsf{empe}(F, x)$. We write simply empe if F and x are understood.

We next introduce formulas $(\!|F'|\!)^*_m$ and $(\!|F'|\!)_m$, which are the building blocks for representing counting star formulas. Formula $(\!|F'|\!)^*_m$ means that F' marked with m and $\mathsf{empEx}_\emptyset(F, x)$ are the only extensions of F that hold in the neighborhood of $x_1, \ldots, x_n$ (F' may hold for *any number* of neighbors). Formula $(\!|F'|\!)_m$ means that F' holds for *exactly one* element in the neighborhood of $x_1, \ldots, x_n$, and all other neighbors have empty extensions. More precisely, let $F' \in \mathsf{exts}(x_1, \ldots, x_n, x)$. Define

$$(\!|F'|\!)^*_m \equiv G_E(x_1, \ldots, x_n) \ \wedge\ \forall x.\, (\textstyle\bigwedge_{i=1}^n x \neq x_i) \Rightarrow (F' \wedge \mathsf{Mark}_m(x)) \vee \mathsf{empEx}_\emptyset(F, x)$$

$$(\!|F'|\!)_m \equiv (\!|F'|\!)^*_m \ \wedge\ \exists^{=1}x.\ \ \textstyle\bigwedge_{i=1}^n x \neq x_i \ \wedge\ F' \wedge \mathsf{Mark}_m(x)$$

where $m \in \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$. Observe that $G \circledast \mathsf{empe}\ \sim\ G$ if $G \equiv (\!|F'|\!)^*_m$ or $G \equiv (\!|F'|\!)_m$ for some F' and m. Also note that $(\!|F'|\!)^*_m \circledast (\!|F'|\!)^*_m \sim (\!|F'|\!)^*_m$.

$$
\begin{array}{ll}
E \land F & -\ \text{EQCAT formula} \\
\quad F & -\ \text{GCCAT formula}
\end{array}
$$

$$
\begin{aligned}
&\mathcal{S}_m\llbracket E \land F \land \exists^{s_1} x.F'_1 \land \ldots \land \exists^{s_k} x.F'_k \rrbracket \ = \\
&= \ E \ \land \ \mathcal{K}\llbracket F \rrbracket \circledast \mathcal{X}_m\llbracket \exists^{s_1} x.F'_1 \rrbracket \circledast \ldots \circledast \mathcal{X}_m\llbracket \exists^{s_k} x.F'_k \rrbracket
\end{aligned}
$$

$$\mathcal{K}\llbracket F \rrbracket \ = \ F \ \land \ (\forall x.\, (\textstyle\bigwedge_{i=1}^{n} x \neq x_i) \Rightarrow \mathsf{empEx}_{\emptyset}(F, x))$$

$$\mathcal{X}_m\llbracket \exists^{0} x.\, F' \rrbracket \quad = \mathsf{empe}$$

$$\mathcal{X}_m\llbracket \exists^{i+1} x.\, F' \rrbracket = \llparenthesis F' \rrparenthesis_m \circledast \mathcal{X}_m\llbracket \exists^{i} x.\, F' \rrbracket$$

$$\mathcal{X}_m\llbracket \exists^{i^+} x.\, F' \rrbracket \ = \mathcal{X}_m\llbracket \exists^{i} x.\, F' \rrbracket \circledast \llparenthesis F' \rrparenthesis^{*}_m$$

Fig. 6. Translation of Counting Stars to Spatial Notation

Translation of counting stars. Figure 6 presents the translation of counting stars to spatial notation. The idea of the translation is to replace $\exists^{=k} x.\, F'$ with the spatial conjunction of k formulas $\llparenthesis F' \rrparenthesis_m \circledast \ldots \circledast \llparenthesis F' \rrparenthesis_m$ where $m \in \{\{1\}, \{2\}\}$. The purpose of the marker m is to ensure that each of the k witnesses for x that are guaranteed to exist by $\llparenthesis F' \rrparenthesis_m \circledast \ldots \circledast \llparenthesis F' \rrparenthesis_m$ are distinct. The reason that the witnesses are distinct for $m \neq \emptyset$ is that no two of them can satisfy $B_i(x)$ at the same time for $i \in m$.

To show the correctness of the translation in Figure 6, define e^m to be the L'-environment obtained by extending L-environment e according to marking m, and $\overline{e_1}$ to be the restriction of an L' environment e_1 to language L. More precisely, if e is an environment in language L, for $m \in \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$, define environment e^m in language L' by 1) $e^m\, r = e\, r$ for $r \in L$ and 2) for $q \in \{1, 2\}$, let $(e\, B_q)\, d = \mathsf{True} \iff q \in m \ \land \ d \notin \{e\, x_1, \ldots, e\, x_n\}$. Conversely, if e_1 is an environment in language L', define environment $\overline{e_1}$ in language L by $\overline{e_1}\, r = e_1\, r$ for all $r \in L$. Lemma 13 below gives the correctness criterion for translation in Figure 6.

Lemma 13. *If e is an environment for language L, C a counting star formula in language L, and $m \in \{\{1\}, \{2\}, \{1, 2\}\}$, then $\llbracket C \rrbracket e = \mathcal{S}_m\llbracket C \rrbracket e^m$.*

$$
\begin{array}{ll}
(1) & \llparenthesis T_1 \rrparenthesis_1 \circledast \llparenthesis T_2 \rrparenthesis_2 \leadsto \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2} \\
(2) & \llparenthesis T_1 \rrparenthesis_1 \circledast \llparenthesis T_2 \rrparenthesis^{*}_2 \leadsto \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2} \circledast \llparenthesis T_2 \rrparenthesis^{*}_2 \\
(3) & \llparenthesis T_1 \rrparenthesis^{*}_1 \circledast \llparenthesis T_2 \rrparenthesis_2 \leadsto \llparenthesis T_1 \rrparenthesis^{*}_1 \circledast \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2} \\
(4) & \llparenthesis T_1 \rrparenthesis^{*}_1 \circledast \llparenthesis T_2 \rrparenthesis^{*}_2 \leadsto \llparenthesis T_1 \rrparenthesis^{*}_1 \circledast \llparenthesis T_2 \rrparenthesis^{*}_2 \circledast \llparenthesis T_1 \odot T_2 \rrparenthesis^{*}_{1,2} \\
(5) & \llparenthesis T \rrparenthesis^{*}_1 \leadsto \mathsf{empe} \\
(6) & \llparenthesis T \rrparenthesis^{*}_2 \leadsto \mathsf{empe}
\end{array}
$$

Fig. 7. Transformation Rules for Combining Spatial Conjuncts

Combining quantifier-free formulas. Let $C_1 \circledast C_2$ be a spatial conjunction of two counting-star formulas

$$\begin{array}{l} C_1 \equiv E \wedge F_1 \wedge \exists^{s_{1,1}} x.F'_{1,1} \wedge \ldots \wedge \exists^{s_{1,k}} x.F'_{1,k} \\ C_2 \equiv E \wedge F_2 \wedge \exists^{s_{2,1}} x.F'_{2,1} \wedge \ldots \wedge \exists^{s_{2,k}} x.F'_{2,l} \end{array}$$

where F_1 and F_2 are GCCAT formulas with $\mathsf{FV}(F_1) = \mathsf{FV}(F_2) = \{x_1, \ldots, x_n\}$, $E \wedge F_1$ and $E \wedge F_2$ are EQCAT formulas, and $E \equiv \bigwedge_{j=1}^{m} y_j = x_{i_j}$.

Note that we assume that the two GCCAT formulas F_1 and F_2 have same free variables and that the equalities E in the two EQCAT formulas are the same. This assumption is justified because either 1) $C_1 \circledast C_2$ make inconsistent assumptions about equalities among $x_1, \ldots, x_n$, and therefore $C_1 \circledast C_2$ is equivalent to False, or 2) $C_1 \circledast C_2$ make same assumptions about equalities among $x_1, \ldots, x_n$, so we can rewrite C_1 and C_2 to satisfy the our assumption by exchanging variables x_i and y_j in the definition of an EQCAT formula.

To show how to transform formula $\mathcal{S}_1[\![C_1]\!] \circledast \mathcal{S}_2[\![C_2]\!]$ into a disjunction of formulas of the form $\mathcal{S}_{1,2}[\![C_3]\!]$, we introduce the following notation. If T is a formula, let $S(T)$ denote the set of positive literals in T_1 that do not contain equality. Let $T_1 \in \mathsf{exts}(F_1, x)$ and $T_2 \in \mathsf{exts}(F_2, x)$. (Note that $\mathsf{exts}(F_1, x) = \mathsf{exts}(F_2, x)$.) We define the partial operation $T_1 \odot T_2$ as follows. The result of $T_1 \odot T_2$ is defined iff $S(T_1) \cap S(T_2) = \emptyset$. If $S(T_1) \cap S(T_2) = \emptyset$, then $T_1 \odot T_2 = T$ where T is the unique element of $\mathsf{exts}(F_1, x)$ such that $S(T) = S(T_1) \cup S(T_2)$. Similarly to $\odot$, we define the partial operation $F_1 \oplus F_2$ for F_1 and F_2 GCCAT formulas with $\mathsf{FV}(F_1) = \mathsf{FV}(F_2) = \{x_1, \ldots, x_n\}$. The result of $F_1 \oplus F_2$ is defined iff $S(F_1) \cap S(F_2) = \emptyset$. If $S(F_1) \cap S(F_2) = \emptyset$, then $F_1 \oplus F_2$ is the unique GCCAT formula F such that $\mathsf{FV}(F) = \{x_1, \ldots, x_n\}$ and $S(F) = S(F_1) \cup S(F_2)$. The following Lemma 14 notes that $\odot$ and $\oplus$ are sound rules for computing spatial conjunction of certain quantifier-free formulas.

Lemma 14. *If $T_1, T_2 \in \mathsf{exts}(x_1, \ldots, x_n, x)$ then $T_1 \circledast T_2 \sim T_1 \odot T_2$. If F_1 and F_2 are GCCAT formulas with $\mathsf{FV}(F_1) = \mathsf{FV}(F_2) = \{x_1, \ldots, x_n\}$, then $F_1 \circledast F_2 \sim F_1 \oplus F_2$.*

Rules for transforming spatial conjuncts. We transform formula $\mathcal{S}_1[\![C_1]\!] \circledast \mathcal{S}_2[\![C_2]\!]$ into a disjunction of formulas of the form $\mathcal{S}_{1,2}[\![C_3]\!]$ as follows.

The first step in transforming $C_1 \circledast C_2$ is to replace $\mathcal{K}[\![F_1]\!] \circledast \mathcal{K}[\![F_2]\!]$ with $\mathcal{K}[\![F_1 \oplus F_2]\!]$ if $F_1 \oplus F_2$ is defined, or False if $F_1 \oplus F_2$ is not defined.

The second step is summarized in Figure 7, which presents rules for combining conjuncts resulting from $\mathcal{X}_1[\![\exists^{s_1}.F_1]\!]$ and $\mathcal{X}_2[\![\exists^{s_2} x.F_2]\!]$ into conjuncts of the form $\mathcal{X}_{1,2}[\![\exists^{s} x.F]\!]$. The intuition is that $(\!|T|\!)^*_m$ and $(\!|T|\!)_m$ represent a finite abstraction of all possible neighborhoods of $x_1, \ldots, x_n$, and the rules in Figure 7 represent the ways in which different portions of the neighborhoods combine using spatial conjunction. We apply the rules in Figure 7 modulo commutativity and associativity of $\circledast$, the fact that emp is a unit for $\circledast$, as well as the idempotence of $(\!|T|\!)^*_m$. Rules $(1)-(4)$ are applicable only when the occurrence of $T_1 \odot T_2$ on the right-hand side of the rule is defined. We apply rules $(1)-(4)$ as long as possible, and then apply rules $(5), (6)$. Moreover, we only allow the sequences of rule applications that eliminate all occurrences of $(\!|T|\!)_1$, $(\!|T|\!)_1^*$, $(\!|T|\!)_2$, $(\!|T|\!)_2^*$, leaving only $(\!|T|\!)_{1,2}$ and $(\!|T|\!)_{1,2}^*$. Note also that the are only finitely many non-equivalent expressions that can be obtained by sequences of applications of rules in Figure 7. Namely, an application of rules (1)–(3) decreases the total number of spatial conjuncts of the form $(\!|T|\!)_1$ and $(\!|T|\!)_2$, multiple applications of rule (4) to the same pair of spatial conjuncts are unnecessary because of the idempotence of $(\!|T_1 \odot T_2|\!)_{1,2}^*$ (so we never perform them), and rules (5), (6) reduce the total number of spatial conjuncts. The following Lemma 15 gives partial correctness of rules in Figure 7.

Lemma 15. *If $G_1 \rightsquigarrow G_2$, then $G_2 \Rightarrow G_1$ is valid.*

Define $G_1 \overset{C}{\Longrightarrow} G_2$ to hold iff both of the following two conditions hold: **1)** G_2 results from G_1 by replacing $\mathcal{K}[\![F_1]\!] \circledast \mathcal{K}[\![F_2]\!]$ with $\mathcal{K}[\![F_1 \oplus F_2]\!]$ if $F_1 \oplus F_2$ is defined, or False if $F_1 \oplus F_2$ is not defined, and then applying some sequence of rules in Figure 7 such that rules $(5), (6)$ are applied only when rules $(1)-(4)$ are not applicable; **2)** G_2 contains only spatial conjuncts of the form $(\!|T|\!)_{1,2}$ and $(\!|T|\!)_{1,2}^*$. From Lemma 15 and Lemma 14 we immediately obtain Lemma 16.

Lemma 16. *If $G_1 \overset{C}{\Longrightarrow} G_2$, then $G_2 \Rightarrow G_1$ is valid.*

The rule for computing the spatial conjunction of counting star formulas is the following. If C_1, C_2, and C_3 are counting star formulas, define $\mathcal{R}(C_1, C_2, C_3)$ to hold iff $\mathcal{S}_1[\![C_1]\!] \circledast \mathcal{S}_2[\![C_2]\!] \overset{C}{\Longrightarrow} \mathcal{S}_{1,2}[\![C_3]\!]$. We compute spatial conjunction by replacing $C_1 \circledast C_2$ with $\bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3$. Our goal is therefore to show the equivalence

$$C_1 \circledast C_2 \quad \sim \quad \bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3 \tag{3}$$

The validity of $\bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3 \;\Rightarrow\; (C_1 \circledast C_2)$ follows from Lemma 16 and Lemma 13.

Lemma 17. $(\bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3) \Rightarrow (C_1 \circledast C_2)$ *is a valid formula for every pair of counting star formulas C_1 and C_2.*

We next consider the converse claim. If $[\![C_1 \circledast C_2]\!]e$, then there are e_1 and e_2 such that $\mathsf{split}\, e\, e_1\, e_2$, $[\![C_1]\!]e_1$, and $[\![C_2]\!]e_2$. By considering the atomic types induced in e, e_1 and e_2 by elements in $D \setminus \{e\, x_1, \ldots, e\, x_n\}$, we construct a sequence of $\rightsquigarrow$ transformations in Figure 7 that convert $\mathcal{S}_1[\![C_1]\!] \circledast \mathcal{S}_2[\![C_2]\!]$ into a formula $\mathcal{S}_{1,2}[\![C_3]\!]$ such that $[\![C_3]\!]e = \mathsf{True}$.

Lemma 18. *$C_1 \circledast C_2 \;\Rightarrow\; \bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3$ is a valid formula for every pair of counting star formulas C_1 and C_2.*

From Lemma 17 and Lemma 18 we obtain the desired Theorem 19, which shows the correctness of our rules for computing spatial conjunction of formulas of quantifier depth at most one.

Theorem 19. *The equivalence (3) holds for every pair of counting star formulas C_1 and C_2.*

8 Further Remarks

In this section we present two additional remarks regarding spatial conjunction. The first remark notes that we must be careful when extracting a subformula from a formula and labelling it with a new predicate. The second remark shows how to encode spatial conjunction in second-order logic, thus providing some insight into the expressive power of spatial conjunction.

8.1 Extracting Subformulas in the Presence of $\circledast$

In two-variable logic with counting C^2 we may efficiently transform formula into an unnested form by introducing new predicate names and naming subformulas using these predicates. This transformations is a standard step in decidability proofs for two-variable logic with counting [22, 45].

The satisfiability of the resulting formula is equivalent to the satisfiability of the original formula. An extraction of a subformula G and its replacement with a new predicate P can be justified by a substitution lemma of the form:

$$[\![F[P := G]]\!]e = [\![F]\!](e[P := [\![G]\!]e])$$

where e is the environment (model). This substitution lemma does not hold in the presence of spatial conjunction that splits the values of newly introduced predicates. Namely,

$$[\![(F_1 \circledast F_2)[P := G]]\!]e \;\Rightarrow\; [\![F_1 \circledast F_2]\!](e[P := [\![G]\!]e])$$

holds, but the converse implication does not hold because the value $[\![G]\!]e$ of the relation P might be split on the right-hand side.

It is therefore interesting to divide predicates into *splittable* and *non-splittable* predicates, and have spatial conjunction split only the interpretations of splittable predicates. The substitution lemma then holds when P is a non-splittable predicate.

Note, however, that in the presence of non-splittable predicates we cannot translate counting stars into spatial notation and thus use unnested form to eliminate all spatial conjunctions from first-order formulas. As a result, adding spatial conjunction of formulas of large quantifier depth to two-variable logic with counting may increase the expressive power of the resulting logic.

We also remark that if the language contains only one splittable unary predicate A_S, then it is easy to simulate the splitting of objects of the universe, which is the semantics of spatial conjunction in [28]. Namely, we use some fixed unary predicate A_0 to denote all “live” objects, and make all quantifiers range only over the objects that satisfy A_0.

8.2 Representing $\circledast$ in Second-Order Logic

In this section we give a simple translation from the first-order logic with spatial conjunction and inductive definitions [27, Chapter 4] to second-order logic. This gives an upper bound on the expressive power of first-order logic with spatial conjunction and inductive definitions.

Consider first-order logic extended with the spatial conjunction $\circledast$ and the least-fixpoint operator. The syntax of the least-fixpoint operator is

$$(\mathsf{lfp}\, P, x_1, \ldots, x_n.F)(y_1, \ldots, y_n)$$

where F is a formula that may contain new free variables $P, x_1, \ldots, x_n$. The meaning of the least-fixpoint operator is that the relation which is the least fixpoint of the monotonic transformation on predicates

$$(\lambda x_1, \ldots, x_n.P(x_1, \ldots, x_n)) \mapsto (\lambda x_1, \ldots, x_n.F)$$

holds for $y_1, \ldots, y_n$. To ensure the monotonicity of the transformation on predicates, we require that P occurs only positively in F.

$$\mathcal{A} = \{A_1, \ldots, A_n\}$$

$$\mathcal{F} = \{f_1, \ldots, f_m\}$$

$$\begin{aligned}
[\![F' \circledast F'']\!] = \exists\, & A'_1, \ldots, A'_n, f'_1, \ldots, f'_m, \\
& A''_1, \ldots, A''_n, f''_1, \ldots, f''_m.\ \mathcal{B}[\![F' \circledast F'']\!]
\end{aligned}$$

$$\begin{aligned}
& \mathcal{B}[\![F' \circledast F'']\!] = \\
& \bigwedge_{i=1}^{n} (\mathsf{split}_1\, A_i\, A'_i\, A''_i) \wedge \bigwedge_{i=1}^{m} (\mathsf{split}_2\, f_i\, f'_i\, f''_i) \wedge \\
& [\![F']\!][A_i := A'_i]_{i=1}^{n}[f_i := f'_i]_{i=1}^{m} \wedge \\
& [\![F'']\!][A_i := A''_i]_{i=1}^{n}[f_i := f''_i]_{i=1}^{m}
\end{aligned}$$

$$\begin{aligned}
\mathsf{split}_1\, A\, A'\, A'' \equiv \forall x.\ & (A(x) \Leftrightarrow (A'(x) \vee A''(x))) \wedge \\
& \neg(A'(x) \wedge A''(x))
\end{aligned}$$

$$\begin{aligned}
\mathsf{split}_2\, f\, f'\, f'' \equiv \forall x\, y.\ & (f(x,y) \Leftrightarrow (f'(x,y) \vee f''(x,y))) \wedge \\
& \neg(f'(x,y) \wedge f''(x,y))
\end{aligned}$$

$$\begin{aligned}
& [\![(\mathsf{lfp}\, P, x_1, \ldots, x_n.F)(y_1, \ldots, y_n)]\!] = \\
& \quad \forall P. (\forall x_1, \ldots, x_n. (F \Leftrightarrow P(x_1, \ldots, x_n))) \Rightarrow P(y_1, \ldots, y_n)
\end{aligned}$$

Fig. 8. Translation of Spatial Conjunction and Inductive Definitions into Second-Order Logic

Figure 8 presents the translation from first-order logic extended with spatial conjunction and least-fixpoint operator to second-order logic. The translation directly mimics the semantics of $\circledast$ and lfp.

In second-order logic, the relations in $L = \mathcal{A} \cup \mathcal{F}$ become free variables.

To translate $\circledast$, use second-order quantification to assert the existence of new unary and binary relations that partition the relations in L into relations in L' and L''. Then perform a syntactic replacement of relations in L with the corresponding relations in L' for the first formula, and with the corresponding relations in L'' for the second formula.

Translating lfp is also straightforward. The property that P is a fixpoint of F is easily expressible. To encode that $y_1, \ldots, y_n$ hold for the *least* fixpoint of F, we state that $y_1, \ldots, y_n$ hold for all fixpoints of F, using universal second-order quantification over P.

We also note that the translation of $\circledast$ in Figure 8 uses only existential second-order quantification, which points to another class of formulas where spatial conjunction can be eliminated if we are only concerned with satisfiability. Namely, if F' and F'' are first-order formulas (without $\circledast$ or lfp), then $F' \circledast F''$ is satisfiable iff the first-order formula $\mathcal{B}[\![F' \circledast F'']\!]$ in the extended language is satisfiable. As a slight generalization, define the following class of "interesting" formulas:

1. a first-order formula F is an interesting formula;
2. if F_1 and F_2 are interesting formulas, so is $F_1 \circledast F_2$;
3. if F_1 and F_2 are interesting formulas, so is $F_1 \vee F_2$

The satisfiability of each interesting formula is equivalent to the satisfiability of the corresponding first-order formula in an extended vocabulary. In particular, the satisfiability of the class of formulas formed starting from formulas in two-variable logic with counting and applying only $\vee$ and $\circledast$ is decidable.

9 Further Related Work

Records have been studied in the context of functional and object-oriented programming languages [11, 14, 23, 29, 42, 46–48, 57]. The main difference between existing record notations and our system is that the interpretation of a record in our system is a predicate on an object, where an object is linked to other objects forming a graph, as opposed to being a type that denotes a value (with values typically representable as finite trees). Our view is appropriate for programming languages such as Java and ML that can manipulate structures using destructive updates. Our generalizations allow the developers to express both incoming and outgoing references of objects, and allow the developers to express typestate changes.

We have developed role logic to provide a foundation for role analysis [30–33]. We have subsequently studied a simplification of role analysis constraints and showed a characterization of such constraints using formulas [34, 35]. Multifields and multislots are present already in [32, Section 8.1]. In this section we have shown that role logic provides a unifying framework for all these constraints and goes beyond them in 1) being closed under the fundamental boolean logical operations, and, 2) being closed under spatial conjunction for an interesting class of formulas. The view of roles as predicates is equivalent to the view of roles as sets and works well in the presence of data abstraction [39,40].

The parametric analysis based on there-valued logic was introduced in [53, 54]. Other approaches to verifying shape invariants include [13,19–21,26,41]. A decidable logic for expressing connectivity properties of the heap was presented in [4]. We use spatial conjunction from separation logic that has been used for reasoning about the heap [7,8,28,51,52]. Description logics [1,6] share many of the properties of role logic and have been traditionally applied to knowledge bases. [9,10] present doubly-exponential deterministic algorithms for reasoning about the satisfiability of expressive description logics over all structures and over finite structures. The decidability of two-variable logic with counting C^2 was shown in [22], whereas [45] establishes the NEXPTIME-complexity of the satisfiability problem for the fragment C_1^2 with counting up to one.

10 Conclusions

We have shown how to add notation for records to two-variable role logic while preserving its decidability. The resulting notation supports a generalization of traditional records with record specifications that are closed under all boolean operations as well as record concatenation, allow the description of typestate properties, support inverse records, and capture the distinction between open and closed records. We believe that such an expressive and decidable notation is useful as an annotation language used with program analyses and type systems.

Acknowledgements. We thank the participants of the Dagstuhl Seminar 03101 “Reasoning about Shape” for useful discussions on separation logic and shape analysis.

References

1. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.
2. Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram K. Rajamani. Automatic predicate abstraction of C programs. In *Proc. ACM PLDI*, 2001.
3. Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Relative completeness of abstraction refinement for software model checking. In *TACAS'02*, volume 2280 of *LNCS*, page 158, 2002.
4. Michael Benedikt, Thomas Reps, and Mooly Sagiv. A decidable logic for linked data structures. In *Proc. 8th ESOP*, 1999.
5. Lars Birkedal, Noah Torp-Smith, and John C. Reynolds. Local reasoning about a copying garbage collector. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 220–231. ACM Press, 2004.
6. Alexander Borgida. Description logics in data management. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):671–682, 1995.
7. Cristiano Calcagno, Luca Cardelli, and Andrew D. Gordon. Deciding validity in a spatial logic for trees. In *ACM TLDI'02*, 2002.

8. Cristiano Calcagno, Samin Ishtiaq, and Peter W. O'Hearn. Semantic analysis of pointer aliasing, allocation and disposal in hoare logic. In *Proc. 2nd International Conference on Principles and Practice of Declarative Programming*, 2000.
9. Diego Calvanese. Finite model reasoning in description logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 292–303. Morgan Kaufmann, 1996.
10. Diego Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.* PhD thesis, Dipartimento di Informatica e Sistemistica, Universita di Roma "La Sapienza", 1996.
11. Luca Cardelli and John C. Mitchell. Operations on records. In *Theoretical Aspects of Object-Oriented Programming.* The MIT Press, Cambridge, Mass., 1994.
12. C. C. Chang and H. J. Keisler. *Model Theory.* North Holland, 1990.
13. David R. Chase, Mark Wegman, and F. Kenneth Zadeck. Analysis of pointers and structures. In *Proc. ACM PLDI*, 1990.
14. David R. Cheriton and Michael E. Wolf. Extensions for multi-module records in conventional programming languages. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 296–306. ACM Press, 1987.
15. Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th POPL*, 1977.
16. Robert DeLine and Manuel Fähndrich. Enforcing high-level protocols in low-level software. In *Proc. ACM PLDI*, 2001.
17. Robert DeLine and Manuel Fähndrich. Typestates for objects. In *18th ECOOP*, 2004.
18. Manuel Fähndrich and K. Rustan M. Leino. Declaring and checking non-null types in an object-oriented language. In *OOPSLA'03*, 2003.
19. Pascal Fradet and Daniel Le Métayer. Structured gamma. *Science of Computer Programming, SCP, 31(2-3), pp. 263-289*, 1998.
20. Pascal Fradet and Daniel Le Métayery. Shape types. In *Proc. 24th ACM POPL*, 1997.
21. Rakesh Ghiya and Laurie J. Hendren. Putting pointer analysis to work. In *Proc. 25th ACM POPL*, 1998.
22. Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science LICS '97, Warschau*, 1997.
23. Robert Harper and Benjamin Pierce. A record calculus based on symmetric concatenation. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 131–142, Orlando, Florida, 1991.
24. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In *31st POPL*, 2004.
25. Wilfrid Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 1993.
26. Joseph Hummel, Laurie J. Hendren, and Alexandru Nicolau. A general data dependence test for dynamic, pointer-based data structures. In *Proc. ACM PLDI*, 1994.
27. Neil Immerman. *Descriptive Complexity.* Springer-Verlag, 1998.
28. Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. 28th ACM POPL*, 2001.
29. Mark Jones and Simon Peyton Jones. Lightweight extensible records for Haskell. In *Haskell Workshop*, 1999.
30. Viktor Kuncak. Designing an algorithm for role analysis. Master's thesis, MIT Laboratory for Computer Science, 2001.
31. Viktor Kuncak, Patrick Lam, and Martin Rinard. A language for role specifications. In *Proceedings of the 14th Workshop on Languages and Compilers for Parallel Computing*, volume 2624 of *Lecture Notes in Computer Science, Springer*, 2001.
32. Viktor Kuncak, Patrick Lam, and Martin Rinard. Roles are really great! Technical Report 822, Laboratory for Computer Science, Massachusetts Institute of Technology, 2001.

33. Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Proc. 29th POPL*, 2002.
34. Viktor Kuncak and Martin Rinard. Typestate checking and regular graph constraints. Technical Report 863, MIT Laboratory for Computer Science, 2002.
35. Viktor Kuncak and Martin Rinard. Existential heap abstraction entailment is undecidable. In *10th Annual International Static Analysis Symposium (SAS 2003)*, San Diego, California, June 11-13 2003.
36. Viktor Kuncak and Martin Rinard. On role logic. Technical Report 925, MIT CSAIL, 2003.
37. Viktor Kuncak and Martin Rinard. On the boolean algebra of shape analysis constraints. Technical report, MIT CSAIL, August 2003.
38. Viktor Kuncak and Martin Rinard. Boolean algebra of shape analysis constraints. In *Proc. 5th International Conference on Verification, Model Checking and Abstract Interpretation*, 2004.
39. Patrick Lam, Viktor Kuncak, and Martin Rinard. On modular pluggable analyses using set interfaces. Technical Report 933, MIT CSAIL, December 2003.
40. Patrick Lam, Viktor Kuncak, and Martin Rinard. Generalized typestate checking using set interfaces and pluggable analyses. *SIGPLAN Notices*, 2004. to appear.
41. Anders Møller and Michael I. Schwartzbach. The Pointer Assertion Logic Engine. In *Proc. ACM PLDI*, 2001.
42. Wolfgang Naraschewski and Markus Wenzel. Object-oriented verification based on record subtyping in higher-order logic. In *11th TPHOLs*, volume 1479 of *LNCS*, pages 349–366, 1998.
43. Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proc. CSL, Paris 2001*, volume 2142 of *LNCS*, 2001.
44. Martin Otto. *Bounded Variable Logics and Counting: A Study in Finite Models*. Lecture Notes in Logic 9. Springer, 1997.
45. Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. on Computing*, 29(4):1083–1117, 2000.
46. Francois Pottier. A constraint-based presentation and generalization of rows. In *Eighteenth Annual IEEE Symposium on Logic In Computer Science (LICS'03)*, Ottawa, Canada, June 2003. To appear.
47. Didier Remy. Typechecking records and variants in a natural extension of ml. In *POPL*, pages 77–88, 1989.
48. Didier Remy. Typing record concatenation for free. In *POPL*, pages 166–176, 1992.
49. Thomas Reps, Mooly Sagiv, and Alexey Loginov. Finite differencing of logical formulas for static analysis. In *Proc. 12th ESOP*, 2003.
50. Thomas Reps, Mooly Sagiv, and Greta Yorsh. Symbolic implementation of the best transformer. In *Proc. 5th International Conference on Verification, Model Checking and Abstract Interpretation*, 2004.
51. John C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Proceedings of the Symposium in Celebration of the Work of C.A.R. Hoare*, 2000.
52. John C. Reynolds. Separation logic: a logic for shared mutable data structures. In *17th LICS*, pages 55–74, 2002.
53. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. In *Proc. 26th ACM POPL*, 1999.
54. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.
55. Robert E. Strom and Daniel M. Yellin. Extending typestate checking using conditional liveness analysis. *IEEE Transactions on Software Engineering*, May 1993.
56. Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, January 1986.
57. Mitchell Wand. Type inference for record concatenation and multiple inheritance. *Information and Computation*, 93(1):1–15, 1991.

58. Eran Yahav and Ganesan Ramalingam. Verifying safety properties using separation and heterogeneous abstractions. In *PLDI*, 2004.
59. Greta Yorsh, Thomas Reps, and Mooly Sagiv. Symbolically computing most-precise abstract operations for shape analysis. In *10th TACAS*, 2004.

A Appendix: Correctness of Spatial Conjunction Elimination

Proposition 8. Every quantifier-free formula F such that $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$ is equivalent to a disjunction of CAT formulas C such that $\mathsf{FV}(C) = \{x_1, \ldots, x_n\}$.

Proof. Let F be a quantifier-free formula and $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$. Transform F to disjunctive normal form F'. Let C be a conjunction in F'. If C contains a literal and its negation, then C is contradictory and we eliminate C from F'. Assume all conjunctions are non-contradictory, and let C be one conjunction. If there exists an atomic formula F_A in variables $\{x_1, \ldots, x_n\}$ such that $F_A \notin C$ and $(\neg F_A) \notin C$, then replace C with the disjunction

$$(C \wedge F_A) \vee (C \wedge \neg F_A)$$

By repeating this process, we obtain a disjunction of CAT formulas.

Lemma 9. Every CAT formula F is either contradictory, or is equivalent to an EQCAT formula F' such that $\mathsf{FV}(F') = \mathsf{FV}(F)$.

Proof. Let F be a CAT formula. If $x_i \neq x_i$ occurs in F, then F is contradictory. If $x_i = x_j$ occurs in F for $i \not\equiv j$, then in all conjuncts other than $x_i = x_j$ replace all occurrences of x_i with x_j. Repeat this process as long as it is possible. Suppose that the resulting formula was not established to be contradictory. Let $y_1, \ldots, y_m$ be variables that occur only on the left-hand side of some equality $y_j = x_{i_j}$. Removing all equalities of the form $y_j = y_j$ yields an EQCAT formula.

Proposition 10. Every quantifier-free formula F such that $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$ can be written as a disjunction of EQCAT formulas C such that $\mathsf{FV}(C) = \{x_1, \ldots, x_n\}$.

Proof. Let F be a quantifier-free formula such that $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$. Using Proposition 8, transform F to disjunction of CAT formulas F_1. Then, for each conjunct C of F_1 apply Lemma 9 to transform C to an EQCAT formula.

Proposition 12. Let F be a formula of such that F has quantifier depth at most one, F has counting degree at most k, and $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$. Then F is equivalent to a disjunction of k-counting-star formulas F_C where $\mathsf{FV}(F_C) = \{x_1, \ldots, x_n\}$.

Proof. Let F be a formula of such that F has quantifier depth at most one, F has counting degree at most k, and $\mathsf{FV}(F) \subseteq \{x_1, \ldots, x_n\}$. Then F is a boolean combination of 1) atomic formulas and 2) formulas of the form $\exists^s z. F'$ where F' is quantifier-free and $\mathsf{FV}(F') = \{z, x_1, \ldots, x_n\}$. Because z is a bound variable, rename it to x in each formula F'. Let F_1 be the result of transforming this boolean combination to disjunctive normal form. Consider a disjunct C of F_1. As in the proof of Proposition 10, and treating quantified formulas as atomic syntactic entities, transform C into disjunction of formulas of the form

$$\bigwedge_{j=1}^{m} y_j = w_{i_j} \ \wedge \ F \ \wedge \bigwedge_{F' \in S} (\exists^{\beta(F')} x.\, F')^{\alpha(F')}$$

where $\beta(F') \in C_{k+1}$, $\alpha(F') \in \{0,1\}$ for $F' \in S$, and where $\bigwedge_{j=1}^{m} y_j = w_{i_j} \wedge F$ is an EQCAT formula with $y_1, \ldots, y_m, w_1, \ldots, w_p$ distinct variables such that $\{y_1, \ldots, y_m, w_1, \ldots, w_p\} = \{x_1, \ldots, x_n\}$, and $\mathsf{FV}(F') \subseteq \{x, x_1, \ldots, x_n\}$ for $F' \in S$. Here S is the set of formulas of the form $\exists^{\beta(F')} x.\, F'$ that end up conjoined with the EQCAT formula as the result of transformation to normal form. By replacing each y_j with w_{i_j} in each F', enforce that $\mathsf{FV}(F') \subseteq \{x, w_1, \ldots, w_p\}$. Using Proposition 10, transform each F' to a disjunction of EQCAT formulas. By applying the equivalences

$$\begin{aligned} \exists^{\geq k_1} x.\, \bigvee_{i=1}^{q} B_i \ &\sim \bigvee_{\sum_{j=1}^{q} l_j = k_1} \ \bigwedge_{i=1}^{q} \exists^{\geq l_i} x.\, B_i \\ \exists^{= k_1} x.\, \bigvee_{i=1}^{q} B_i \ &\sim \bigvee_{\sum_{j=1}^{q} l_j = k_1} \ \bigwedge_{i=1}^{q} \exists^{= l_i} x.\, B_i \end{aligned}$$

for $B_1, \ldots, B_q$ mutually exclusive, and propagating the disjunction to the top level, ensure that every F' is an EQCAT formula. Then transform each term $(\exists^{\beta(F')} x.\, F')^{\alpha(F')}$ into positive boolean combination of formulas of one of the forms $\exists^{=i} x.\, F'$ for $0 \leq i \leq k$ and $\exists^{\geq k+1} x.\, F'$, using the properties

$$\begin{aligned} \neg \exists^{\geq k_1} x.\, F' \ &\sim \bigvee_{i=0}^{k_1 - 1} \exists^{=i} x.\, F' \\ \neg \exists^{= k_1} x.\, F' \ &\sim \bigvee_{i \in \{0, \ldots, k\} \setminus \{k_1\}} \exists^{=i} x.\, F' \ \vee \exists^{\geq k+1} x.\, F' \end{aligned}$$

Next ensure that each F' is not merely an EQCAT, but in fact a GCCAT such that $F' \in \mathsf{exts}(F, x)$, as follows.

Suppose that F' contains a literal L_1 complementary to some literal occurring in GCCAT formula F. If L_1 occurs in $\exists^{=i} x.\, F'$ for $i > 0$ or in $\exists^{\geq k+1} x.\, F'$, then the entire conjunct is contradictory and we eliminate it. If L_1 occurs in $\exists^{=0} x.\, F'$, then $\exists^{=0} x.\, F'$ is implied by F, so eliminate it. Assume that F' has no literals complementary to literals in F. Then F' contains $w_i \neq w_j$ for all $i \not\equiv j$. Next ensure that $x \neq w_i$ is a conjunct for $1 \leq i \leq p$, as follows. Suppose that F' contains the conjunct $x = w_i$ for some $1 \leq i \leq p$.

There is clearly at most one interpretation of x that is equal to interpretation of w_i, so if $\beta(F') \in \{2, 3, \ldots, k, (k+1)^{+}\}$ then F and F' are contradictory and the entire conjunction is False, so assume $\beta(F') \in \{0, 1\}$. For the same reason, $\exists^{=1} x.\, F'$ is equivalent to $\exists x. F'$, so if $\beta(F') = 1$, then replace x with w_i in

F' giving a GCCAT formula F'' such that $\mathsf{FV}(F'') = \mathsf{FV}(F)$. By definition of GCCAT formulas, either F and F'' are equivalent, so $F \land (\exists x.F'') \sim F$, or F and F'' are contradictory, and the entire conjunction is False.

Assume therefore that $x \neq w_i$ occurs in F' for all $1 \leq i \leq p$. This means that F' is a GCCAT formula. Because $\mathsf{FV}(F') = \{x, w_1, \ldots, w_p\}$ and F' does not contain a literal complementary to a literal from F, eliminating from F' atomic formulas that occur in F yields an element of $\mathsf{exts}(F, x)$.

To ensure that there exists exactly one conjunct of the form $\exists^s x.\, F'$ for each $F' \in \mathsf{exts}(F, x)$, use the fact that the $k+1$ formulas $\exists^{=i} x.\, F'$, for $0 \leq i \leq k$, and $\exists^{\geq k+1} x.\, F'$ form a partition (they are mutually exclusive and their disjunction is True).

Lemma 13. If e is an environment for language L, C a counting star formula in language L, and $m \in \{\{1\},\{2\},\{1,2\}\}$, then $\llbracket C \rrbracket e = \mathcal{S}_m \llbracket C \rrbracket e^m$.

Proof. Formula E contains only equalities, so $\llbracket E \rrbracket e$ iff $\llbracket E \rrbracket e^m$. It therefore suffices to show that

$$\llbracket \mathcal{K}\llbracket F \rrbracket \circledast \mathcal{X}_m \llbracket \exists^{s_1} x.F_1' \rrbracket \circledast \ldots \circledast \mathcal{X}_m \llbracket \exists^{s_k} x.F_k' \rrbracket \rrbracket e^m = \mathsf{True} \tag{4}$$

iff $\llbracket F \rrbracket e = \mathsf{True}$ and for all i, $\llbracket \exists^{s_i} x.F_i' \rrbracket e = \mathsf{True}$.

$\Rightarrow$): Let (4) hold. Then there exist $e_0, e_1, \ldots, e_k$ such that $\mathsf{split}\, e^m [e_0\, e_1\, \ldots e_k]$, $\llbracket \mathcal{K}\llbracket F \rrbracket \rrbracket e_0 = \mathsf{True}$, and $\llbracket \mathcal{X}_m \llbracket \exists^{s_i} x.F_i' \rrbracket \rrbracket e_i = \mathsf{True}$ for $1 \leq i \leq k$.

We first show $\llbracket F \rrbracket e = \mathsf{True}$. Note first that $\llbracket G_E \rrbracket e_i = \mathsf{True}$ for $1 \leq i \leq k$. Namely, because both $(\!|F'|\!)^*_m$ and $(\!|F'|\!)_m$ entail G_E, so does $\mathcal{X}_m \llbracket \exists^{s_i} x.F_i' \rrbracket$, by definition of $\mathcal{X}_m \llbracket \rrbracket$ and split. Therefore, e_0 is the only environment among $e_0, e_1, \ldots, e_k$ that may have non-empty relations between the elements interpreting $x_1, \ldots, x_n$. As a result, $\llbracket F \rrbracket e^m = \llbracket F \rrbracket e_0$. But $\llbracket F \rrbracket e_0 = \mathsf{True}$ because $\llbracket \mathcal{K}\llbracket F \rrbracket \rrbracket e_0 = \mathsf{True}$. Therefore $\llbracket F \rrbracket e^m = \mathsf{True}$, and F contains no symbols from $L' \setminus L$, so $\llbracket F \rrbracket e = \mathsf{True}$.

We next show $\llbracket \exists^{s_i} x.F_i' \rrbracket e = \mathsf{True}$ for $1 \leq i \leq k$. For $s_i = p^+$, from $\llbracket \mathcal{X}_m \llbracket \exists^{s_i} x.F_i' \rrbracket \rrbracket e_i = \mathsf{True}$ we have that there exist $e_{i,0}, e_{i,1}, \ldots, e_{i,p}$ such that 1) $\mathsf{split}\, e_i [e_{i,0}, e_{i,1}, \ldots, e_{i,p}]$, 2) $\llbracket (\!|F'|\!)^*_m \rrbracket e_{i,0} = \mathsf{True}$, and 3) $\llbracket (\!|F'|\!)_m \rrbracket e_{i,j} = \mathsf{True}$ for $1 \leq j \leq p$. Similarly, for $s_i < p$, we have that there exist $e_{i,1}, \ldots, e_{i,s_i}$ such that 1) $\mathsf{split}\, e_i [e_{i,1}, \ldots, e_{i,s_i}]$, and 2) $\llbracket (\!|F'|\!)_m \rrbracket e_{i,j} = \mathsf{True}$ for $1 \leq j \leq s_i$. Note that whenever $\llbracket (\!|F'|\!)^*_m \rrbracket e_{i,j}$ or $\llbracket (\!|F'|\!)_m \rrbracket e_{i,j}$ holds, we can split elements of the domain D into two disjoint sets: elements $E_{i,j}$ for which $\mathsf{empEx}_\emptyset(F, x)$ holds, and elements $N_{i,j}$ for which $F' \land \mathsf{Mark}_m(x)$ holds. If $\llbracket (\!|F'|\!)_m \rrbracket e_{i,j}$, then $|N_{i,j}| = 1$, by definition of $\llbracket (\!|F'|\!)_m \rrbracket e_{i,j}$. Moreover, by definition of split and because $m \neq \emptyset$, we have $N_{i_1,j_1} \cap N_{i_2,j_2} = \emptyset$ for $\langle i_1, j_1 \rangle \neq \langle i_2, j_2 \rangle$. Observe that, for a given domain element $d \in D$, the atomic type extension corresponding to e^m with $x \mapsto d$ is the union of atomic type extensions corresponding to each $e_{i,j}$. The atomic type extension for d in $e_{i,j}$ is either $F' \land \mathsf{Mark}_m(x)$, or $\mathsf{empEx}_\emptyset(F, x)$. Therefore, the atomic type extension for d in e^m is either $F' \land \mathsf{Mark}_m(x)$ if $d \in N_{i,j}$ for some i, j, or $\mathsf{empEx}_\emptyset(F, x)$ if for all i, j, $d \notin N_{i,j}$. If $N_i = \{d \mid \llbracket F_i' \rrbracket e^m [x \mapsto d] = \mathsf{True}\}$, then $N_i = \biguplus_j N_{i,j}$. If $s_i = k < p$ then $|N_i| = \sum_{j=1}^{s_i} |N_{i,j}| = \sum_{i=j}^{s_i} 1 = s_i$, so $\llbracket \exists^{=k} x.\, F_i' \rrbracket e^m = \mathsf{True}$. Because $\exists^{=k} x.\, F_i'$

is formula in language L, we have $\llbracket \exists^{=k} x.\, F_i' \rrbracket e = \mathsf{True}$. Similarly, if $s_i = p^+$, then $|N_i| = |N_{i,0}| + \sum_{j=1}^{p} |N_{i,j}| = |N_{i,0}| + p \geq p$, so $\llbracket \exists^{\geq k} x.\, F_i' \rrbracket e^m = \mathsf{True}$ and therefore $\llbracket \exists^{\geq p} x.\, F_i' \rrbracket e = \mathsf{True}$. In both cases, $\llbracket \exists^{s_i} x.\, F_i' \rrbracket e = \mathsf{True}$.

This completes one direction of the implication, we next show the converse direction.

$\Leftarrow$): Let $\llbracket F \rrbracket e = \mathsf{True}$ and for all i where $1 \leq i \leq k$, $\llbracket \exists^{s_i} x. F_i' \rrbracket e = \mathsf{True}$. We construct environments $e_0, e_1, \ldots, e_k$ such that 1) $\mathsf{split}\, e^m\, [e_0, e_1, \ldots, e_k]$ 2) $\llbracket \mathcal{K}\llbracket F \rrbracket \rrbracket e_0 = \mathsf{True}$, and 3) $\llbracket \mathcal{X}_m \llbracket \exists^{s_i} x. F_i' \rrbracket \rrbracket e_i = \mathsf{True}$ for all i where $1 \leq i \leq k$. We construct $e_0, e_1, \ldots, e_k$ by assigning the tuples of relations in e to one of the environments $e_0, e_1, \ldots, e_k$, as follows. We only need to decide on splitting the tuples $\langle d_1, \ldots, d_q \rangle$ where all but one value $d_1, \ldots, d_q$ are from the set $D_X = \{ex_1, \ldots, ex_n\}$, the values of relations on other tuples do not affect the truth value of formulas in question and can be split arbitrarily. If $\{d_1, \ldots, d_q\} \subseteq D_X$, then we assign the tuple to e_0, as a result, $\llbracket \mathcal{K}\llbracket F \rrbracket \rrbracket e_0 = \mathsf{True}$. If $\{d_1, \ldots, d_q\} \backslash D_X = \{d\}$, then let i be such that F_i' is the unique extension of F with the property $\llbracket F_i' \rrbracket e[x \mapsto d] = \mathsf{True}$. Then assign the tuple $\langle d_1, \ldots, d_q \rangle$ to the environment e_i and also assign the values $(e\, B_l)\, d$ for all $l \in m$ to e_i. Because we assign each relevant tuple to exactly one e_i, we ensure $\mathsf{split}\, e^m\, [e_0, e_1, \ldots, e_k]$. Let $D_E = \{d \mid \llbracket F_i' \rrbracket e[x \mapsto d] = \mathsf{True}\}$, then also $D_E = \{d \mid \llbracket F_i' \rrbracket e_i[x \mapsto d] = \mathsf{True}\}$. Because $\llbracket \exists^{s_i} x. F_i' \rrbracket e = \mathsf{True}$, $|D_E| = s_i$ for $s_i < p$ and $|D_E| \geq p$ for $s_i = p^+$. Let $s_i < p$. Then split e_i into $e_{i,1}, \ldots, e_{i,s_i}$ by assigning exactly one element $d \in D_E$ to one $e_{i,j}$. When assigning an element we assign the values of all relations from L, as well as the relations B_1 and B_2. This ensures that $\llbracket \llparenthesis F_i' \rrparenthesis_m \rrbracket e_{i,j} = \mathsf{True}$ for all $1 \leq i \leq s_i$. For $s_i = p^+$, we split e_i into $e_{i,0}, e_{i,1}, \ldots, e_{i,p}$ by assigning exactly one element to each of $e_{i,1}, \ldots, e_{i,p}$ and assigning the remaining elements to $e_{i,0}$. In both cases, we obtain $\llbracket \mathcal{X}_m \llbracket \exists^{s_i} x. F_i' \rrbracket \rrbracket e_i = \mathsf{True}$.

Lemma 15. If $G_1 \rightsquigarrow G_2$, then $G_2 \Rightarrow G_1$ is valid.

Proof. We show the claim for each of the rules (1)–(6).

Rule (1): Let $T_1 \odot T_2$ be defined and let $\llbracket \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2} \rrbracket e = \mathsf{True}$ for an L'-environment e. Let $d \in D$ be the unique domain element such that $\llbracket T_1 \odot T_2 \rrbracket e[x \mapsto d] = \mathsf{True}$. Let e_1 and e_2 be such that $\mathsf{split}\, e\, [e_1, e_2]$, $\llbracket T_1 \rrbracket e_1[x \mapsto d] = \mathsf{True}$ and $\llbracket T_2 \rrbracket e_2[x \mapsto d] = \mathsf{True}$, and $e_p B_q d = \mathsf{True}$ iff $p = q$ for $p, q \in \{1, 2\}$. In other words, e_1 and e_2 split e by assigning tuples validating T_1 to e_1, tuples validating T_2 to e_2, and by assigning B_1 to e_1 and B_2 to e_2 on the element d. The values of relations $e\, r$ containing tuples with an element $d' \notin \{ex_1, \ldots, ex_n, d\}$ are all False, because $\llbracket \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2} \rrbracket e = \mathsf{True}$, so we let the values of $e_1 r$ and $e_2 r$ for those tuples also be empty. Then d is the only element outside $\{ex_1, \ldots, ex_n\}$ such that $\llbracket T_1 \rrbracket e_1[x \mapsto d] = \mathsf{True}$, and d is also the only element outside $\{ex_1, \ldots, ex_n\}$ such that $\llbracket T_2 \rrbracket e_2[x \mapsto d] = \mathsf{True}$. As a result, $\llbracket \llparenthesis T_1 \rrparenthesis_1 \rrbracket e_1 = \mathsf{True}$ and $\llbracket \llparenthesis T_2 \rrparenthesis_2 \rrbracket e_2 = \mathsf{True}$, so $\llbracket \llparenthesis T_1 \rrparenthesis_1 \circledast \llparenthesis T_2 \rrparenthesis_2 \rrbracket e = \mathsf{True}$.

To show the claim for rules (2), (3), (4), we proceed similarly as for rule (1).

Rule (2): Let $T_1 \odot T_2$ be defined and let $\llbracket \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2} \circledast \llparenthesis T_2 \rrparenthesis_2^* \rrbracket e = \mathsf{True}$. Then there are e' and e'' such that $\mathsf{split}\, e\, [e', e'']$, $\llbracket \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2} \rrbracket e' = \mathsf{True}$ and $\llbracket \llparenthesis T_2 \rrparenthesis_2^* \rrbracket e'' = \mathsf{True}$. Let d be the unique element such that $\llbracket T_1 \odot T_2 \rrbracket e'[x \mapsto$

$d] = \mathsf{True}$, and let $d_1, \ldots, d_k$ be the list of all (distinct) elements such that $\llbracket \llparenthesis T_2 \rrparenthesis_2^* \rrbracket e''[x \mapsto d_i] = \mathsf{True}$. Note that $d \notin \{d_1, \ldots, d_k\}$, because $e' B_2 d = \mathsf{True}$, $e'' B_2 d_i = \mathsf{True}$ for all $1 \leq i \leq k$, and $\mathsf{split}\, e\, [e', e'']$. We construct e_1 and e_2 such that $\mathsf{split}\, e\, [e_1, e_2]$ as follows. We assign B_1, as well as the values of relations that hold according to T_1 on element d to e_1, and we assign B_2, as well as the values of relations that hold according to T_2 on element d to e_2. We assign B_2 as well as the values of relations that hold according to T_2 on $d_1, \ldots, d_k$ to e_2. The values of B_1 and the relations on $d_1, \ldots, d_k$ for e_1 are empty. For such e_1 and e_2 we have $\llbracket \llparenthesis T_1 \rrparenthesis_1 \rrbracket e_1 = \mathsf{True}$ and $\llbracket \llparenthesis T_2 \rrparenthesis_2^* \rrbracket e_2 = \mathsf{True}$, so $\llbracket \llparenthesis T_1 \rrparenthesis_1 \circledast \llparenthesis T_2 \rrparenthesis_2^* \rrbracket e = \mathsf{True}$.

Rule (3) is analogous to rule (2).

Rule (4): Let $T_1 \odot T_2$ be defined and let $\llbracket \llparenthesis T_1 \rrparenthesis_1^* \circledast \llparenthesis T_2 \rrparenthesis_2^* \circledast \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2}^* \rrbracket = \mathsf{True}$. Then there are e',e'', e''' such that $\mathsf{split}\, e\, [e', e'', e''']$, $\llbracket \llparenthesis T_1 \rrparenthesis_1^* \rrbracket e' = \mathsf{True}$, $\llbracket \llparenthesis T_2 \rrparenthesis_2^* \rrbracket e'' = \mathsf{True}$, and $\llbracket \llparenthesis T_1 \odot T_2 \rrparenthesis_{1,2}^* \rrbracket e''' = \mathsf{True}$. Then there are three sets of elements N', N'', N''', where N' contains elements that validate T_1 in e', N'' contains elements that validate T_2 in e'', and N''' contains elements that validate $T_1 \odot T_2$ in e'''. We have $N' \cap N''' = \emptyset$ and $N'' \cap N''' = \emptyset$, whereas $N' \cap N''$ need not be empty. Each element $d \notin \{ex_1, \ldots, ex_n\}$ validates in e either 1) $\mathsf{empEx}_{\emptyset}(F, x)$, if $d \notin N' \cup N'' \cup N'''$, or 2) T_1, if $d \in N' \setminus N''$, or 3) T_2, if $d \in N'' \setminus N'$, or 4) $T_1 \odot T_2$, if $d \in (N' \cap N'') \cup N'''$. We construct environments e_1,e_2,e_3 by assigning B_1 and relations from T_1 to elements in $N' \setminus N''$ to e_1, assigning B_2 and elements in $N' \setminus N''$ to e_2, and splitting relations on elements in $(N' \cap N'') \cup N'''$ into those for T_1, which we assign to e_1, and those for T_2, which we assign to e_2. We then have $\llbracket \llparenthesis T_1 \rrparenthesis_1^* \rrbracket e_1 = \mathsf{True}$ and $\llbracket \llparenthesis T_2 \rrparenthesis_2^* \rrbracket e_2 = \mathsf{True}$, so $\llbracket \llparenthesis T_1 \rrparenthesis_1^* \circledast \llparenthesis T_2 \rrparenthesis_2^* \rrbracket = \mathsf{True}$.

Rules (5), (6): Directly from the definitions of empe and $\llparenthesis F' \rrparenthesis_m^*$ it follows that $\mathsf{empe} \Rightarrow \llparenthesis F' \rrparenthesis_m^*$.

Lemma 17. $(\bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3) \Rightarrow (C_1 \circledast C_2)$ is a valid formula for every pair of counting star formulas C_1 and C_2.

Proof. Let $\llbracket \bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3 \rrbracket e$ hold for some L-environment e. Then $\llbracket C_3 \rrbracket e = \mathsf{True}$ for some C_3 such that $\mathcal{S}_1 \llbracket C_1 \rrbracket \circledast \mathcal{S}_2 \llbracket C_2 \rrbracket \overset{C}{\Longrightarrow} \mathcal{S}_{1,2} \llbracket C_3 \rrbracket$. By Lemma 16, $\mathcal{S}_{1,2} \llbracket C_3 \rrbracket \Rightarrow \mathcal{S}_1 \llbracket C_1 \rrbracket \circledast \mathcal{S}_2 \llbracket C_2 \rrbracket$ is valid. By Lemma 13 and $\llbracket C_3 \rrbracket e = \mathsf{True}$, we have $\llbracket \mathcal{S}_{1,2} \llbracket C_3 \rrbracket \rrbracket e^{1,2} = \mathsf{True}$. Therefore, $\llbracket \mathcal{S}_1 \llbracket C_1 \rrbracket \circledast \mathcal{S}_2 \llbracket C_2 \rrbracket \rrbracket e^{1,2} = \mathsf{True}$. This means that there are e_1 and e_2 such that $\mathsf{split}\, e^{1,2}\, [e_1, e_2]$, $\llbracket \mathcal{S}_1 \llbracket C_1 \rrbracket \rrbracket e_1 = \mathsf{True}$, and $\llbracket \mathcal{S}_2 \llbracket C_2 \rrbracket \rrbracket e_2 = \mathsf{True}$. From Lemma 13 we have $\llbracket C_1 \rrbracket \overline{e_1} = \mathsf{True}$, and $\llbracket C_2 \rrbracket \overline{e_2} = \mathsf{True}$. From $\mathsf{split}\, e^{1,2}\, [e_1, e_2]$ it follows that $\mathsf{split}\, e\, [\overline{e_1}, \overline{e_2}]$, so $\llbracket C_1 \circledast C_2 \rrbracket e = \mathsf{True}$.

Lemma 18. $C_1 \circledast C_2 \Rightarrow \bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3$ is a valid formula for every pair of counting star formulas C_1 and C_2.

Proof. Let $\llbracket C_1 \circledast C_2 \rrbracket e = \mathsf{True}$ for some L-environment e. Then there are e_1 and e_2 such that $\mathsf{split}\, e\, [e_1, e_2]$, $\llbracket C_1 \rrbracket e_1 = \mathsf{True}$ and $\llbracket C_2 \rrbracket e_2 = \mathsf{True}$. By Lemma 13, $\mathcal{S}_1 \llbracket C_1 \rrbracket e_1^1 = \mathsf{True}$ and $\mathcal{S}_2 \llbracket C_2 \rrbracket e_2^2 = \mathsf{True}$. We construct $\mathcal{S}_{1,2} \llbracket C_3 \rrbracket$ such that $\mathcal{S}_1 \llbracket C_1 \rrbracket \circledast \mathcal{S}_2 \llbracket C_2 \rrbracket \overset{C}{\Longrightarrow} \mathcal{S}_{1,2} \llbracket C_3 \rrbracket$ and $\llbracket C_3 \rrbracket e = \mathsf{True}$, as follows.

Let K_1 be the GCCAT part of C_1 and let K_2 be the GCCAT part of C_2. Let $D_X = D \backslash \{e\, x_1, \ldots, e\, x_n\}$. For each $d \in D_X$, let T_1^d be the type extension induced by d in e_1, that is, let $T_1^d \in \mathsf{exts}(K_1, x)$ be the formula such that $\llbracket T_1^d \rrbracket e_1^1[x \mapsto d] =$

True. Similarly, let $T_2^d \in \mathsf{exts}(K_2, x)$ be the formula such that $\llbracket T_2^d \rrbracket e_2^2[x \mapsto d] = \mathsf{True}$. Because $\mathsf{split}\, e\, [e_1, e_2]$, the operation $T_1 \odot T_2$ is defined and $\llbracket T_1 \odot T_2 \rrbracket e^{1,2}[x \mapsto d] = \mathsf{True}$. Because $\mathcal{S}_1\llbracket C_1 \rrbracket e_1^1 = \mathsf{True}$, with each d we can associate an occurrence $\mu_1(d)$ in $\mathcal{S}_1\llbracket C_1 \rrbracket$ of a formula $F_{\mu_1(d)}$ where $F_{\mu_1(d)}$ is of the form $\llparenthesis T_1^d \rrparenthesis_1$ or of the form $\llparenthesis T_1^d \rrparenthesis_1^*$, and an environment $e_{1,\mu_1(d)}$ such that $\mathsf{split}\, e_1^1\, [e_{1,0}, (e_{1,\mu_1(d)})_{\mu(d)}]$, such that $\mathcal{K}\llbracket K_1 \rrbracket e_{1,0} = \mathsf{True}$, and such that for every d, $\llbracket F_{\mu_1(d)} \rrbracket e_{1,\mu_1(d)} = \mathsf{True}$. Analogously, for each d we can associate an occurrence $\mu_2(d)$ in $\mathcal{S}_2\llbracket C_2 \rrbracket$ of a formula $F_{\mu_2(d)}$ of the form $\llparenthesis T_2^d \rrparenthesis_2$ or of the form $\llparenthesis T_2^d \rrparenthesis_2^*$, and an environment $e_{2,\mu_2(d)}$ such that $\mathsf{split}\, e_2^2\, [e_{2,0}, (e_{2,\mu_2(d)})_{\mu_2(d)}]$, such that $\mathcal{K}\llbracket K_2 \rrbracket e_{2,0} = \mathsf{True}$, and such that for every d, $\llbracket F_{\mu_2(d)} \rrbracket e_{2,\mu_2(d)} = \mathsf{True}$.

We compute C_3 by first combining $\mathcal{K}\llbracket K_1 \rrbracket$ and $\mathcal{K}\llbracket K_2 \rrbracket$ into $\mathcal{K}\llbracket K_1 \oplus K_2 \rrbracket$. From $\mathsf{split}\, e\, [e_1, e_2]$ we conclude that the operation $F_1 \oplus F_2$ is well-defined and that $\llbracket \mathcal{K}\llbracket F_1 \oplus F_2 \rrbracket \rrbracket e_0^{1,2} = \mathsf{True}$ where $e_0^{1,2}$ is given by $\mathsf{split}\, e_0^{1,2}\, [e_{1,0}, e_{2,0}]$.

We next apply rules (1)–(4) in Figure 7, as follows:

1. apply rule (1) once to each pair of occurrences $\mu_1(d)$ and $\mu_2(d)$ if they are of the form $\llparenthesis T_1^d \rrparenthesis_1$ and $\llparenthesis T_2^d \rrparenthesis_2$, respectively; let $\mu(d)$ be the occurrence of the resulting formula $F_{\mu(d)} \equiv \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2}$;
2. apply rule (2) once to each pair of occurrences $\mu_1(d)$ and $\mu_2(d)$ if $\mu_1(d)$ is an occurrence of the form $\llparenthesis T_1^d \rrparenthesis_1$ and $\mu_2(d)$ is an occurrence of the form $\llparenthesis T_2^d \rrparenthesis_2^*$; let $\mu(d)$ be the occurrence of the formula $F_{\mu(d)} \equiv \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2}$ obtained as one of the results;
3. apply rule (3) once to each pair of occurrences $\mu_1(d)$ and $\mu_2(d)$ if $\mu_1(d)$ is an occurrence of the form $\llparenthesis T_1^d \rrparenthesis_1^*$ and $\mu_2(d)$ is an occurrence of the form $\llparenthesis T_2^d \rrparenthesis_2$; let $\mu(d)$ be the occurrence of the formula $F_{\mu(d)} \equiv \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2}$ obtained as one of the results;
4. apply rule (4) once for each pair of occurrences of formulas of the form $\llparenthesis T_1^d \rrparenthesis_1^*$ and $\llparenthesis T_2^d \rrparenthesis_2^*$; for each d such that $\mu_1(d)$ is an occurrence of $\llparenthesis T_1^d \rrparenthesis_1^*$ and $\mu_2(d)$ is an occurrence of $\llparenthesis T_2^d \rrparenthesis_2^*$, let $\mu(d)$ be the occurrence of the resulting formula $F_{\mu(d)} \equiv \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2}^*$.

Note that no rule is applied twice to a distinct pair of occurrences of formulas. This means that the number of applications of rules is uniformly bounded, despite the fact that there is no bound on the size of the model e. In particular, there is no bound on the number of elements d covered by a single application of rule (4). Each formula of the form $\llparenthesis T \rrparenthesis_1$ is $F_{\mu_1(d)}$ for some d and each formula of the form $\llparenthesis T \rrparenthesis_2$ is $F_{\mu_2(d)}$ for some d, and all such formulas are consumed by applications of rules (1)–(3), so the resulting formula has no subformulas of the form $\llparenthesis T \rrparenthesis_1$ or $\llparenthesis T \rrparenthesis_2$. After applying rules (1)–(4), apply rules (5) and (6) to all applicable formulas. The resulting formula F_R has no occurrences of $\llparenthesis T \rrparenthesis_1^*$ or $\llparenthesis T \rrparenthesis_2^*$ either, it contains only occurrences of formulas of forms $\llparenthesis T \rrparenthesis_{1,2}$ and $\llparenthesis T \rrparenthesis_{1,2}^*$.

For each of the finitely many occurrences $\mu(d)$ in F_R we construct $e_{\mu(d)}^{1,2}$, splitting $e^{1,2}$ into the environment $e_0^{1,2}$ defined above, and the environments $e_{\mu(d)}^{1,2}$, by assigning the type extension of d in $e^{1,2}$ to $e_{\mu(d)}^{1,2}$. By construction, $\mathsf{split}\, e^{1,2}\, [e_0^{1,2}, (e_{\mu(d)}^{1,2})_{\mu(d)}]$. To show $\llbracket F_R \rrbracket e^{1,2} = \mathsf{True}$, it suffices to show

$$\llbracket F_c \rrbracket e_c^{1,2} = \mathsf{True} \tag{5}$$

for every occurrence $c = \mu(d_0)$. Fix an occurrence c, and let $\delta = \{d \mid \mu(d) = c\}$. By definition of $e_c^{1,2}$, the type extension induced by each $d \in \delta$ in $e_c^{1,2}$ is $T_1^d \odot T_2^d$, and the type extension of each $d \in D_X \setminus \delta$ is an empty extension. Therefore, $\llbracket \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2}^* \rrbracket e_c^{1,2} = \mathsf{True}$. If $F_c \equiv \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2}^*$ then the equation (5) already holds. If $F_c \equiv \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2}$, then F_c was generated by one of the rules (1)–(3), which means that δ is a singleton set. Namely, if F_c was generated by rules (1) or (2), then there is exactly one d such that $\mu_1(d) = c$, namely d_0, and similarly if F_c was generated by rule (3), then there is exactly one d such that $\mu_2(d) = c$, again d_0. In both cases, $\delta = \{d_0\}$, so d_0 is the unique d with type extension $T_1^d \odot T_2^d$, which means that $\llbracket \llparenthesis T_1^d \odot T_2^d \rrparenthesis_{1,2} \rrbracket e_c^{1,2} = \mathsf{True}$ and the equation (5) holds.

We finally apply idempotence to ensure that no $\llparenthesis T \rrparenthesis_m^*$ occurs more than once. The resulting formula F'_R is equivalent to F_R, so $\llbracket F'_R \rrbracket e^{1,2} = \mathsf{True}$, F'_R is of the form $\mathcal{S}_{1,2}\llbracket C_3 \rrbracket$, and $\mathcal{S}_1\llbracket C_1 \rrbracket \circledast \mathcal{S}_2\llbracket C_2 \rrbracket \overset{C}{\Longrightarrow} \mathcal{S}_{1,2}\llbracket C_3 \rrbracket$. From $\mathcal{S}_{1,2}\llbracket C_3 \rrbracket$ we recover C_3 using the inverse of the translation in Figure 6. By Lemma 13 we have $\llbracket C_3 \rrbracket e = \mathsf{True}$, completing the proof.