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

Abstract. Role logic is 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.
In this paper, we show how to use role logic to describe open and closed records, as well as 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 the 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

1 Introduction

In [22] we introduced *role logic*, a notation for describing properties of relational structures that arise in shape analysis, databases and knowledge bases. The role logic notation aims to combine the simplicity of role declarations [19] and the well-established first-order logic. The use of implicit arguments and syntactic sugar of role logic supports easy and concise expression of common idioms for describing data structures with mutable references and makes role logic attractive as a generalization of type systems in imperative languages, without sacrificing the expressiveness of a specification language based on first-order logic.

The decidability properties of role logic make it appropriate for communicating information to static analysis tools that go beyond simple type checkers. In [22, Section 4] we establish the decidability of the fragment RL^2 of role logic by

⋆ This research was supported in part by DARPA Contract F33615-00-C-1692, NSF Grant CCR00-86154, NSF Grant CCR00-63513, and the Singapore-MIT Alliance.

⋆⋆ SAS 2004, Verona, Italy, 26–28 August exhibiting a correspondence with two-variable logic with counting C^2, which was shown decidable in [12]. The fragment RL^2 is closed under all boolean operations, generalizes boolean shape analysis constraints [23] of shape analysis [34, 38] and generalizes the non-transitive constraints of role analysis [19].

Generalized records in role logic. In this paper we give a systematic account of the field and slot declarations of role analysis [19] 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 [10, 35] and more general changes in relationships between objects such as movements of objects between data structures [19, 34]. **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 [16, 33] 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, 16]. 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 [12] 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 of objects 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 [19, 22, 26]. By selecting a finite-height lattice of properties for a given program fragment, abstract interpretation [9] can be used to synthesize properties of objects at intermediate program points [2,3,14,19,34,37,39]. Decidability and closure properties of our notation are essential for the completeness and predictability of the resulting static analysis [24].

Outline and contributions. Section 2 reviews the syntax and the semantics of role logic [22]. Section 3 defines spatial conjunction in role logic and identifies its novel use: describing record concatenation. Sections 4 and 5 show how to use spatial conjunction in role logic to describe a generalization of records. These generalizations are useful for expressing properties of objects and memory cells in imperative programs. Section 6 demonstrates that our notation is a generalization of local constraints arising in role analysis [19] 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. As a result, we obtain a decidable notation for generalized records that supports record concatenation.

2 A Decidable Two-Variable Role Logic RL^2

Figure 1 presents the two-variable role logic RL^2 [22]. We proved in [22] that RL^2 has the same expressive power as the 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 combinations of formulas of the form $\exists^{\geq k}x.\, F(x)$. The logic C^2 was shown decidable in [12] and the complexity for the C_1^2 fragment of C^2 (with counting up to one) was established in [30]. 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.

In [22] we show how to perform static analysis with RL^2 by observing that straight-line code with procedure invocations can be encoded in RL^2. When loop invariants and procedure specifications are expressed in RL^2, the resulting verification conditions belong to RL^2 and can be discharged using a decision procedure. The analysis of sequences of non-deterministic actions, such as partially specified procedure calls, is possible because RL^2 has a decision procedure that is parametric with respect to the vocabulary of sets and relations, which means that the intermediate program states can be modelled by introducing a fresh

$$
\begin{array}{rcl}
F & ::= & A \mid f \mid \mathsf{EQ} \mid F_1 \wedge F_2 \mid \neg F \mid F' \mid {\sim}F \mid \mathsf{card}^{\geq k} F \\
d \in D & - & \text{domain of first-order structure (set of all objects)} \\
A \in \mathcal{A} & - & \text{unary predicates (sets)} \\
f \in \mathcal{F} & - & \text{binary predicates (relations)} \\
e & :: & (\{1,2\} \to D) \cup (\mathcal{A} \to D \to \mathsf{bool}) \cup (\mathcal{F} \to D^2 \to \mathsf{bool}) \\
[\![A]\!]e & = & e\,A(e\,1) \qquad\qquad\qquad [\![f]\!]e = e\,f(e\,2, e\,1) \\
[\![\mathsf{EQ}]\!]e & = & (e\,2) = (e\,1) \\
[\![F_1 \wedge F_2]\!]e & = & ([\![F_1]\!]e) \wedge ([\![F_2]\!]e) \qquad [\![\neg F]\!]e = \neg([\![F]\!]e) \\
[\![F']\!]e & = & [\![F]\!](e[1 \mapsto (e\,2)]) \qquad [\![{\sim}F]\!]e = [\![F]\!](e[1 \mapsto (e\,2), 2 \mapsto (e\,1)]) \\
[\![\mathsf{card}^{\geq k} F]\!]e & = & |\{d \in D \mid [\![F]\!](e[1 \mapsto d, 2 \mapsto (e\,1)])\}| \geq k \\
F_1 \vee F_2 & \equiv & \neg(\neg F_1 \wedge \neg F_2) \qquad\qquad F_1 \Rightarrow F_2 \;\equiv\; \neg F_1 \vee F_2
\end{array}
$$

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

copy of the state vocabulary for each program point. Moreover, given a family of abstraction predicates [34] expressible in RL^2, the techniques of [24, 39] can be used to synthesize loop invariants.

In this paper, we focus on the use of role logic to describe generalized records. The results of this paper further demonstrate the expressive power of RL^2, and the appropriateness of RL^2 as the foundation of both the constraints supplied by the developer, and the constraints synthesized by a static analysis.

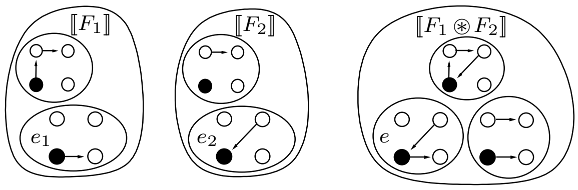
3 Spatial Conjunction

This section introduces our notion 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.

Example 1. The formula $[f \Rightarrow A]$ is true for an object whose every f-field points to an A object, the formula $[g \Rightarrow B]$ means that every g-field points to a B object, so $[f \Rightarrow A] \wedge [g \Rightarrow B]$ denotes the objects that have 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 the 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 $P_f \equiv (\mathsf{card}^{=1} f) \wedge (\mathsf{card}^{=0}(g \vee h))$ means that the object has only one outgoing f-edge and no other edges. The formula $P_g \equiv$



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

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

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

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

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

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

priority: $\wedge$ binds strongest, then $\circledast$, then $\vee$; $\quad F \ \approx\ G$ means $\forall e.\, \llbracket F \rrbracket e = \llbracket G \rrbracket e$

$$(F_1 \circledast F_2) \circledast F_3 \ \approx\ F_1 \circledast (F_2 \circledast F_3) \qquad F \circledast \mathsf{emp} \ \approx\ \mathsf{emp} \circledast F \ \approx\ F$$

$$F_1 \circledast F_2 \ \approx\ F_2 \circledast F_1 \qquad F_1 \circledast (F_2 \vee F_3) \ \approx\ F_1 \circledast F_2 \ \vee\ F_1 \circledast F_3$$

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

$(\mathsf{card}^{=1} g) \wedge (\mathsf{card}^{=0}(f \vee h))$ means that the object has only one outgoing g-edge and no other edges. If we "physically join" the two records, each of which has one field, we obtain a record that has two fields, and is described by the formula $P_{fg} \ \equiv\ (\mathsf{card}^{=1} f) \wedge (\mathsf{card}^{=1} g) \wedge (\mathsf{card}^{=0} h)$. Note that it is *not* the case that $P_{fg} \ \approx\ P_f \wedge P_g$. In fact, 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 the 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. The idea of splitting is that each unary relation $(e\, A)$ is a disjoint union of relations $(e_1\, A)$ and $(e_2\, A)$, and similarly each binary relation $(e\, f)$ is a disjoint union of relations $(e_1\, f)$ and $(e_2\, f)$. For $\mathsf{split}\, e\, [e_1\, e_2]$ we also require that the domain D of objects is the same in all of e_1, e_2, and e. If we consider models e as graphs, then our notion of spatial conjunction keeps a fixed set of nodes, and splits the edges of the graph[1], as illustrated in Figure 2. The notion of splitting generalizes to splitting into any number of environments. Having introduced spatial conjunction $\circledast$, we observe that for P_f, P_g, and P_{fg} of Example 2, we simply have $P_{fg} = P_f \circledast P_g$.

[1] See [22, Page 6] for a comparison of our notion of spatial conjunction with [16].

4 Field Complement

As a step towards a 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 3. 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 relation D^2, whereas $g \vee h \vee i$ is the union of all relations other than f.

To address the notational problem illustrated in Example 3, we introduce the symbol edges, which denotes the union of all binary relations, formally $\mathsf{edges} \equiv \bigvee_g g$, and the notation $-f$ (*field complement* of f), which denotes the union of all relations other than f, formally $-f \equiv \bigvee_{g \neq f} g$. This additional notation allows us to avoid explicitly listing all fields in the language when stating properties like P_f. Formula P_f from Example 3 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 $P_f \wedge \mathsf{card}^{=0}{\sim}\mathsf{edges}$.

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 and inverse records.

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

Fig. 3. Record Notation

The notation for records and inverse records is presented in Figure 3. 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. We abbreviate $f \xrightarrow{=1} A$ by $f \rightarrow A$ and $A \xleftarrow{=1} f$ by $A \leftarrow f$.

Example 4. 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 $P_3 \equiv P_2 \wedge \mathsf{True} \xleftarrow{*} h$. 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 $P_4 \equiv P_2 \wedge C \leftarrow h$. 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 [18, 19]. 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 [20, 21]. 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. [2]

Roles describing complete sets of fields and slots. Figure 4 shows the translation of role constraints [19, 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.

Simultaneous Roles. In object-oriented programs, objects may participate in multiple data structures. The idea of simultaneous roles [19, 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, then it satisfies the conjunction

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

of predicates. Figure 5 presents the translation of field and slot constraints of simultaneous roles into role logic. Whereas the roles of [19, Section 3] translate to closed records and closed inverse records, the simultaneous roles of [19, Section 7.2] translate specifications that are closer to open records and open inverse records.

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

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

Fig. 4. Translation of Role Constraints [19] into Role Logic Formulas

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

$$\mathcal{O}[\![\texttt{fields } f_1 : S_1, \ldots, f_n : S_n]\!] = \mathcal{C}[\![\texttt{fields } f_1 : S_1, \ldots, f_n : S_n]\!] \circledast \mathsf{card}^{=0}(\textstyle\bigvee_{i=1}^{n} f_i)$$

$$\mathcal{O}[\![g_1, \ldots, g_m \ \texttt{slots } S_1.f_1, \ldots, S_n.f_n]\!] = \mathcal{C}[\![\texttt{slots } S_1.f_1, \ldots, S_n.f_n]\!] \circledast \mathsf{card}^{=0}(\textstyle\bigvee_{i=1}^{m} \sim g_i)$$

Fig. 5. Translation of Simultaneous Role Constraints [19, Section 7.2] into Role Logic Formulas.

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 for a large class of record specifications using the remaining logical operations.

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 [12], and showing that the spatial conjunction of counting stars is equivalent to a disjunction of counting stars. It follows that adding $\circledast$ to (full first-order or two-variable) logic with counting does not change the expressive power of that logic, provided that the operands of $\circledast$ 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 [22], 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 [25] for proof details and additional remarks.

7.1 Atomic Type Formulas

In this section we introduce classes of formulas that correspond to the model-theoretic notion of atomic type [29, Page 20]. We then introduce formulas that describe the notion of counting stars [12, 30]. We conclude this section with Proposition 9, 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*. From the disjunctive normal form theorem for propositional logic, we obtain the following Proposition 5.

Proposition 5. *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. A *general-case CAT (GCCAT)* formula describes the case where all variables denote distinct values: a 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 $i \equiv j$. 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 6. *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 5 and Lemma 6, we obtain the following Proposition 7.

Proposition 7. *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 an integer, let p^+ be a new symbol representing 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 $\exists^c x.\, P$ we mean $\exists^{=c} x.\, P$ if c is an integer, and $\exists^{\geq p} x.\, P$ if $c = p^+$. 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 $c \in C_{p+1}$. A counting star formula describes the neighborhood of an object by specifying an approximation of the number of objects x that realize each extension.

Definition 8 (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 8, 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 $F' \in \mathsf{exts}(F, x)$.

Proposition 9 (Depth-One Normal Form). *Let F be a formula 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 9, 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 $\circledast$ with respect to $\vee$, we have $F_1 \circledast F_2 \;\approx\; (\bigvee_{i=1}^{n_1} C_{1,i}) \circledast (\bigvee_{j=1}^{n_2} C_{2,j}) \;\approx\; \bigvee_{i=1}^{n_1} \bigvee_{j=1}^{n_2} C_{1,i} \circledast C_{2,j}$. 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 $x = x$ occurs in a negated literal. Call F' an *empty extension* and denote it $\mathsf{empEx}(x_1, \ldots, x_n, x)$.

To compute a spatial conjunction of counting star 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

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

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 $G \circledast \mathsf{empe} \;\approx\; G$ if $G \equiv (\!|F'|\!)^*_m$ or $G \equiv (\!|F'|\!)_m$ for some F' and m. Also note that $(\!|F'|\!)^*_m \circledast (\!|F'|\!)^*_m \sim (\!|F'|\!)^*_m$.

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

$$
\begin{array}{l}
E \wedge F - \text{EQCAT formula}, \qquad F - \text{GCCAT formula} \\
\mathcal{S}_m[\![E \wedge F \wedge \exists^{s_1} x.F'_1 \wedge \ldots \wedge \exists^{s_k} x.F'_k]\!] = \\
\qquad = E \wedge \mathcal{K}[\![F]\!] \circledast \mathcal{X}_m[\![\exists^{s_1} x.F'_1]\!] \circledast \ldots \circledast \mathcal{X}_m[\![\exists^{s_k} x.F'_k]\!] \\
\mathcal{K}[\![F]\!] = F \wedge (\forall x. (\bigwedge_{i=1}^{n} x \neq x_i) \Rightarrow \mathsf{empEx}_{\emptyset}(F, x)) \\
\mathcal{X}_m[\![\exists^0 x. F']\!] = \mathsf{empe} \qquad \mathcal{X}_m[\![\exists^{i+1} x. F']\!] = (\!|F'|\!)_m \circledast \mathcal{X}_m[\![\exists^i x. F']\!] \\
\mathcal{X}_m[\![\exists^{i^+} x. F']\!] = \mathcal{X}_m[\![\exists^i x. F']\!] \circledast (\!|F'|\!)^*_m
\end{array}
$$

Fig. 6. Translation of Counting Stars to Spatial Notation

the spatial conjunction of k formulas $(\!|F'|\!)_m \circledast \ldots \circledast (\!|F'|\!)_m$ 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 $(\!|F'|\!)_m \circledast \ldots \circledast (\!|F'|\!)_m$ 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 the L-environment e according to marking m, and $\overline{e_1}$ to be the restriction of the L'-environment e_1 to the language L. More precisely, if e is an L-environment, for $m \in \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$, define the L'-environment e^m by 1) $e^m\, r = e\, r$ for $r \in L$ and 2) for $q \in \{1,2\}$, let $(e\, B_q)\, d = \mathsf{True} \iff q \in m \ \wedge \ d \notin \{e\, x_1, \ldots, e\, x_n\}$. Conversely, if e_1 is an L'-environment, define the L-environment $\overline{e_1}$ by $\overline{e_1}\, r = e_1\, r$ for all $r \in L$. Lemma 10 below gives the correctness criterion for the translation in Figure 6.

Lemma 10. *If e is an L-environment, C a counting star formula in L, and $m \in \{\{1\}, \{2\}, \{1,2\}\}$, then $[\![C]\!]e = \mathcal{S}_m[\![C]\!]e^m$.*

(1) $(\!|T_1|\!)_1 \circledast (\!|T_2|\!)_2 \rightsquigarrow (\!|T_1 \odot T_2|\!)_{1,2}$
(2) $(\!|T_1|\!)_1 \circledast (\!|T_2|\!)^*_2 \rightsquigarrow (\!|T_1 \odot T_2|\!)_{1,2} \circledast (\!|T_2|\!)^*_2$
(3) $(\!|T_1|\!)^*_1 \circledast (\!|T_2|\!)_2 \rightsquigarrow (\!|T_1|\!)^*_1 \circledast (\!|T_1 \odot T_2|\!)_{1,2}$
(4) $(\!|T_1|\!)^*_1 \circledast (\!|T_2|\!)^*_2 \rightsquigarrow (\!|T_1|\!)^*_1 \circledast (\!|T_2|\!)^*_2 \circledast (\!|T_1 \odot T_2|\!)^*_{1,2}$
(5) $(\!|T|\!)^*_1 \rightsquigarrow \mathsf{empe}$
(6) $(\!|T|\!)^*_2 \rightsquigarrow \mathsf{empe}$

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 $E \equiv \bigwedge_{j=1}^{m} y_j = x_{i_j}$. To show how to transform the 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 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 11 notes that $\odot$ and $\oplus$ are sound rules for computing spatial conjunction of certain quantifier-free formulas.

Lemma 11. *If $T_1, T_2 \in \mathsf{exts}(x_1, \ldots, x_n, x)$ then $T_1 \circledast T_2 \approx T_1 \odot T_2$. If F_1 and F_2 are GCCAT formulas with $\mathsf{FV}(F_1) = \mathsf{FV}(F_2) = \{x_1, \ldots, x_n\}$, then $F_1 \circledast F_2 \approx F_1 \oplus F_2$.*

Rules for transforming spatial conjuncts. We transform the 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 $\circledast$, and 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 $(\!|T|\!)^*_{1,2}$. The following Lemma 12 gives the partial correctness of the rules in Figure 7.

Lemma 12. *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 12 and Lemma 11 we immediately obtain Lemma 13.

Lemma 13. *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 $\mathcal{S}_1[\![C_1]\!] \circledast \mathcal{S}_2[\![C_2]\!] \stackrel{C}{\Longrightarrow} \mathcal{S}_{1,2}[\![C_3]\!]$. 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

$$C_1 \circledast C_2 \quad \approx \bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3 \tag{2}$$

The validity of $\bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3 \;\Rightarrow\; (C_1 \circledast C_2)$ follows from Lemma 13 and Lemma 10.

Lemma 14. $(\bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3) \Rightarrow (C_1 \circledast C_2)$ *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 can 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 15. $C_1 \circledast C_2 \;\Rightarrow\; \bigvee_{\mathcal{R}(C_1,C_2,C_3)} C_3$ *is a valid formula for every pair of counting star formulas* C_1 *and* C_2*.*

Theorem 16. *The equivalence (2) holds for every pair of counting star formulas* C_1 *and* C_2*.*

8 Further Related Work

Records have been studied in the context of functional and object-oriented programming languages [7, 13, 17, 31, 32, 36]. 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 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 to allow the developers to express typestate changes.

We have developed role logic to provide a foundation for role analysis [19]. We have subsequently studied a simplification of role analysis constraints and characterized such constraints using formulas [20, 21]. Multifields and multislots are present already in [18, Section 6.1]. In this paper we have shown that role logic provides a unifying framework for all these constraints and goes beyond them in 1) being closed under boolean 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 [27].

The parametric analysis based on three-valued logic is presented in [34]. Other approaches to verifying shape invariants include [8, 11, 15, 28]. 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 [6, 16, 33]. Description logics [1] share many of the properties of role logic and have been traditionally applied to knowledge bases.

9 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. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *Proc. ACM PLDI*, 2001.
3. T. Ball, A. Podelski, and S. K. Rajamani. Relative completeness of abstraction refinement for software model checking. In *TACAS'02*, volume 2280 of *LNCS*, page 158, 2002.
4. M. Benedikt, T. Reps, and M. Sagiv. A decidable logic for linked data structures. In *Proc. 8th ESOP*, 1999.
5. L. Birkedal, N. Torp-Smith, and J. C. Reynolds. Local reasoning about a copying garbage collector. In *31st ACM POPL*, pages 220–231. ACM Press, 2004.
6. C. Calcagno, L. Cardelli, and A. D. Gordon. Deciding validity in a spatial logic for trees. In *ACM TLDI'02*, 2002.
7. L. Cardelli and J. C. Mitchell. Operations on records. In *Theoretical Aspects of Object-Oriented Programming*. The MIT Press, Cambridge, Mass., 1994.
8. D. R. Chase, M. Wegman, and F. K. Zadeck. Analysis of pointers and structures. In *Proc. ACM PLDI*, 1990.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th POPL*, 1977.
10. M. Fähndrich and K. R. M. Leino. Declaring and checking non-null types in an object-oriented language. In *OOPSLA'03*, 2003.
11. P. Fradet and D. L. Métayer. Shape types. In *Proc. 24th ACM POPL*, 1997.
12. E. Grädel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science LICS '97, Warschau*, 1997.
13. R. Harper and B. Pierce. A record calculus based on symmetric concatenation. In *18th ACM POPL*, pages 131–142, Orlando, Florida, 1991.

14. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *31st POPL*, 2004.
15. J. Hummel, L. J. Hendren, and A. Nicolau. A general data dependence test for dynamic, pointer-based data structures. In *Proc. ACM PLDI*, 1994.
16. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. 28th ACM POPL*, 2001.
17. M. Jones and S. P. Jones. Lightweight extensible records for Haskell. In *Haskell Workshop*, 1999.
18. V. Kuncak. Designing an algorithm for role analysis. Master's thesis, MIT Laboratory for Computer Science, 2001.
19. V. Kuncak, P. Lam, and M. Rinard. Role analysis. In *Proc. 29th POPL*, 2002.
20. V. Kuncak and M. Rinard. Typestate checking and regular graph constraints. Technical Report 863, MIT Laboratory for Computer Science, 2002.
21. V. Kuncak and M. Rinard. Existential heap abstraction entailment is undecidable. In *10th Annual International Static Analysis Symposium (SAS 2003)*, San Diego, California, June 11-13 2003.
22. V. Kuncak and M. Rinard. On role logic. Technical Report 925, MIT CSAIL, 2003.
23. V. Kuncak and M. Rinard. On the boolean algebra of shape analysis constraints. Technical report, MIT CSAIL, August 2003.
24. V. Kuncak and M. Rinard. Boolean algebra of shape analysis constraints. In *Proc. 5th International Conference on Verification, Model Checking and Abstract Interpretation*, 2004.
25. V. Kuncak and M. Rinard. On generalized records and spatial conjunction in role logic. Technical Report 942, MIT CSAIL, April 2004.
26. P. Lam, V. Kuncak, and M. Rinard. On modular pluggable analyses using set interfaces. Technical Report 933, MIT CSAIL, December 2003.
27. P. Lam, V. Kuncak, and M. Rinard. Generalized typestate checking using set interfaces and pluggable analyses. *SIGPLAN Notices*, 39:46–55, March 2004.
28. A. Møller and M. I. Schwartzbach. The Pointer Assertion Logic Engine. In *Proc. ACM PLDI*, 2001.
29. M. Otto. *Bounded Variable Logics and Counting: A Study in Finite Models.* Lecture Notes in Logic 9. Springer, 1997.
30. L. Pacholski, W. Szwast, and L. Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. on Computing*, 29(4):1083–1117, 2000.
31. F. Pottier. A constraint-based presentation and generalization of rows. In *18th IEEE LICS*, June 2003.
32. D. Remy. Typing record concatenation for free. In *POPL*, pages 166–176, 1992.
33. J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *17th LICS*, pages 55–74, 2002.
34. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.
35. R. E. Strom and S. Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE TSE*, January 1986.
36. M. Wand. Type inference for record concatenation and multiple inheritance. *Information and Computation*, 93(1):1–15, 1991.
37. E. Yahav and G. Ramalingam. Verifying safety properties using separation and heterogeneous abstractions. In *PLDI*, 2004.
38. G. Yorsh. Logical characterizations of heap abstractions. Master's thesis, Tel-Aviv University, March 2003.
39. G. Yorsh, T. Reps, and M. Sagiv. Symbolically computing most-precise abstract operations for shape analysis. In *10th TACAS*, 2004.