Replace this file with `prentcsmacro.sty` for your meeting, or with `entcsmacro.sty` for your meeting. Both can be found at the ENTCS Macro Home Page.

Quadtrees as an Abstract Domain

Jacob M. Howe[1,4]

Dept of Computing, City University London, UK

Andy King[1,3,5]

School of Computing, University of Kent, Canterbury, UK

Charles Lawrence-Jones[2,6]

School of Computing, University of Kent, Canterbury, UK

Abstract

Quadtrees have proved popular in computer graphics and spatial databases as a way of representing regions in two dimensional space. This hierarchical data-structure is flexible enough to support non-convex and even disconnected regions, therefore it is natural to ask whether this data-structure can form the basis of an abstract domain. This paper explores this question and suggests that quadtrees offer a new approach to weakly relation domains whilst their hierarchical structure naturally lends itself to representation with boolean functions.

Keywords: Weakly relational domains, abstract interpretation, quadtrees, boolean formulae

1 Introduction

Program analyses based on abstract interpretation require an abstract domain. One of the first domains described was that of polyhedra [9] and recent work has investigated subclasses of polyhedra, referred to as weakly relational

[1] This work is supported by EPSRC projects EP/E033105/1 and EP/E034519/1.
[2] Supported by a Nuffield Science Bursary
[3] Partly supported by a Royal Society Industrial Fellowship
[4] Email: jacob@soi.city.ac.uk
[5] Email: a.m.king@kent.ac.uk
[6] Email: clawrencejones@gmail.com

©2010 Published by Elsevier Science B. V.

domains (examples include [6,15,16,17,21]). The motivation for weakly relational domains is the cost of polyhedral domain operations: weakly relational domains restrict the dependencies between variables that can be expressed in order to achieve tractable domain operations whilst retaining sufficient expressivity to be useful.

This paper proposes a new abstract domain based on the well-known data-structure of quadtrees [11]. The domain belongs to the weakly relational domain family, but its representation is not given in terms of linear inequalities. The representation means that disjoint, non-linear and non-convex regions can be represented naturally, but this flexibility comes at a cost.

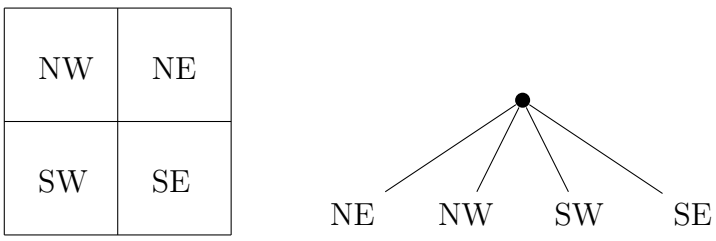
The paper is neutral as to the suitability of quadtrees for use in practical analysers. It is a paper that aspires to promote discussion on the relationship between spatial abstractions and boolean formulae. Nevertheless, the paper makes the following contributions:

- introduces a weakly relational domain for analysis of machine integers that is based on quadtrees
- discusses how this domain might be represented and details how this might be achieved using boolean formulae, either as binary decision diagrams [3] or as formulae in (non-canonical) conjunctive normal form [14]

The paper is structured as follows: section 2 recalls the definition of quadtrees and introduces the underlying idea of using them as an abstract domain; sections 3 and 4 formally introduce the domain and its operations; section 5 discusses the encoding of quadtrees using boolean data-structures and sections 6 and 7 conclude with a survey of related work and a discussion of the strengths and weaknesses of the new domain.

2 Quadtrees

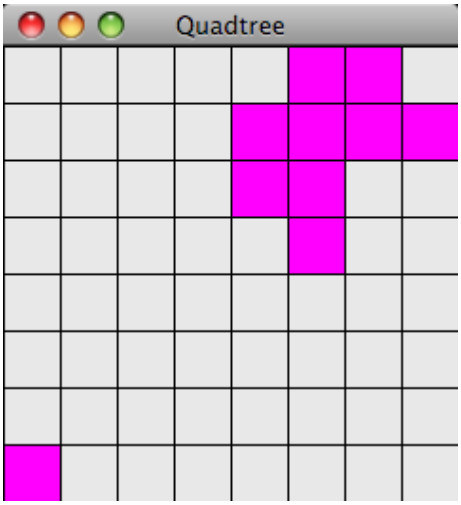
A *quadtree* is a tree where each node has four children; it is interpreted as decomposition of a square in smaller squares, the root being the largest, containing square. A node corresponds to a square and its children to the four squares obtained by dividing the containing square evenly into four. Following [10] the child nodes are ordered anti-clockwise from the top right, as below:



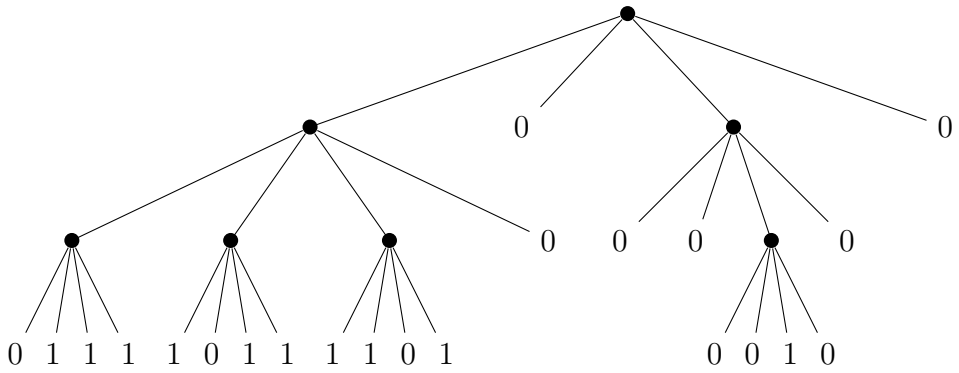
In this work the interest is not only in the decomposition of a square into further squares, but in whether or not these squares are part of some region of interest. Therefore the leaves of quadtrees will be labelled with 0 or 1 to indicate whether or not the corresponding square is part of the region of interest.

Quadtrees are potentially infinite data-structures, as squares can be continually subdivided. However, this work, like others [19], is concerned with analysis over machine integers. This gives a smallest meaningful square, one that is 1×1. Later in this work quadtrees whose smallest square has a larger size will be considered. Henceforth, the minimum square size will be described by the log of its width and this will be referred to as the *granularity* of the quadtree. For example, a quadtree with granularity 2 has minimum square size 4×4. A quadtree with a given granularity is then finite. Assuming that the granularity is a non-negative integer g, a quadtree with $2^n \times 2^n$ root square has leaves at maximum depth $n - g$, where the root is considered to be depth zero.

Consider the following decomposition of an 8×8 grid into 1×1 cells, where the dark cells are the region of interest:



It can be represented by the following quadtree (with granularity 0):



The nature of this decomposition echoes BDDs that have been used to express disjunctive properties [8,13]. This link is further explored in section 5.

3 The Lattice of Quadtrees

This section formally introduces the lattice of quadtrees. The definition introduces quadtrees as purely spatial objects (in fact, divorcing them from their representation as trees) as this provides the most natural description of the lattice.

Quadtrees give a description of two dimensional collections of squares within a regular square grid. Each axis of the grid is intended to capture an analysis variable. There are, of course, likely to be many analysis variables, therefore the domain needs to be able to capture some higher dimensional relationships as well. Although the quadtree domain elements are purely spatial, intuitively, these elements derive from collections of quadtrees, each quadtree in a collection being over a pair of variables. The pairs of variables are not necessarily disjoint, therefore the various quadtrees in a domain element interact via their intersection in higher dimensional space. The domain is weakly relation since the higher dimensional relationships are induced by two dimensional relationships over quadtrees.

3.1 Quadtrees

First, a spatial definition of quadtrees in two dimensions is given. This is then used as the basis of a definition for arbitrary dimensions.

Let $X = \{x_1, ...x_d\}$ be a finite set of variables. Let $I = [min, max) \subset \mathbb{Q}$ denote an interval such that $min, max \in \mathbb{Z}$ and $max = min + 2^n$ for some $n \in \mathbb{N}$. The starting point is the definition of $C_{xy}^{n,i}$, where $x, y \in X$, the set of all squares resulting from the decomposition of the $I \times I$ grid (whose axes are x and y) where the granularity is i. $C_{xy}^{n,i}$ is defined when $i \leq n$ by:

$$
\begin{aligned}
C^{i,i}_{xy} &= \{\{\langle x, y\rangle \mid min \leq x < min + 2^i, min \leq y < min + 2^i\}\} \\
C^{n,i}_{xy} &= \{\{\langle x, y\rangle \mid min \leq x < min + 2^n, min \leq y < min + 2^n\}\} \cup \\
&\quad C^{n-1,i}_{xy} \cup \\
&\quad \{C + \{\langle 0, 2^{n-1}\rangle\} \mid C \in C^{n-1,i}_{xy}\} \cup \\
&\quad \{C + \{\langle 2^{n-1}, 0\rangle\} \mid C \in C^{n-1,i}_{xy}\} \cup \\
&\quad \{C + \{\langle 2^{n-1}, 2^{n-1}\rangle\} \mid C \in C^{n-1,i}_{xy}\}
\end{aligned}
$$

where in the second case $i < n$ and $+$ denotes the Minkowski sum.

Now define $Q^{n,i}_{xy} = \{\cup S \mid S \subseteq C^{n,i}_{xy}\}$. That is, $q_{xy} \in Q^{n,i}_{xy}$ is a subset of $I \times I$ and can be represented by a quadtree. The second superscript, i, will be omitted when it takes the value 0. Note that $Q^{n,i}_{xy}$ could be defined directly, though a recursive formulation might be seen as more natural.

The next definition gives a spatial notion of quadtrees in higher dimensions. Define the projection of a d-dimensional object S onto variables x_j, x_k as follows, $\pi_{jk}(S) = \{\langle a_j, a_k\rangle \mid \langle a_1, ..., a_d\rangle \in S\}$. Now define the expansion of a two dimensional quadtree as $q^+_{x_jx_k} = \cup\{S \mid \pi_{jk}(S) = q_{x_jx_k} \in Q^{n,i}_{x_jx_k}\}$. That is, $q_{xy} \in Q^{n,i}_{xy}$ is interpreted as an n-dimensional, rather than 2 dimensional, object by extending it through the other dimensions, analogous to a prism in three dimensions. Then $Q^{n,i}_X = \{\cap^m_{j=1} q^+_j \mid m \geq 1, q_j \in Q^{n,i}_{x_kx_l}$ for $x_k, x_l \in X, k \neq l\}$. Hence, each $q_X \in Q^{n,i}_X$ is a subset of I^d. Again, the granularity superscript i will be omitted when it is not necessary.

3.2 Meet, Join and Entailment

With the spatial definition of quadtrees, the remaining lattice operations are defined straightforwardly with set operations.

Let the ordering operation on $Q^{n,i}_X$ be defined by $q_1 \models q_2$ iff $q_1 \subseteq q_2$, that is, ordering is by inclusion. Let $\sqcap, \sqcup$ denote the meet and join lattice operations. For $q_1, q_2 \in Q^{n,i}_X$, $q_1 \sqcap q_2 = q_1 \cap q_2$, $q_1 \sqcup q_2 = q_1 \cup q_2$. Notice that $Q^{n,i+1}_X \subset Q^{n,i}_X$.

To conclude, $\langle Q^{n,i}_X, \models, \sqcap, \sqcup\rangle$ forms a finite lattice.

4 Representation and Operations

This section spells out how quadtrees can be represented in terms of their two dimensional projections. It then gives spatial definitions of the domain operations that reduce to operations on each two dimensional projection. However, there are several possibilities as to how a two dimensional quadtree might be realised, and this choice is delayed until the following section.

4.1 Representation

The definition of Q^n_X in section 3 defines elements of the domain as sets of points in I^d without reference to how these sets can be represented. As a weakly relational domain, the expectation is that the representation is in terms of the two variable projections of the space. The definition suggests that each domain element should be represented by a set of quadtrees, $q_{x_jx_k} \in Q^n_{x_jx_k}$.

A domain element $q_X \in Q^n_X$ is represented by a set consisting of exactly one $q_{xy} \in Q^n_{xy}$ for each $x, y \in X$. Such a set, of size $d(d-1)/2$, will be denoted S. Define $[\![S]\!] = \cap\{q^+_{xy} \mid q_{xy} \in S\}$ so as to interpret a set S as a domain element. Note that the same domain element can be represented by different sets.

4.2 Meet

With a set representation for domain elements, meet can be determined pairwise on the individual quadtree components. Over a variable pair, meet is simply intersection: define $q_{xy} \sqcap p_{xy} = q_{xy} \cap p_{xy}$. This lifts to domain elements: where $q_X, p_X \in Q^n_X$, and $q_X = [\![S_q]\!]$, $p_X = [\![S_p]\!]$, meet can be determined by $q_X \sqcap p_X = [\![\{q_{xy} \sqcap p_{xy} | x, y \in X, q_{xy} \in S_q, p_{xy} \in S_p\}]\!]$.

4.3 Variable Elimination

A resolution step tightens a two dimensional quadtree by taking account of the interaction of two others. Where, $q_{xy} \in Q^n_{xy}, q_{yz} \in Q^n_{yz}$, $res(q_{xy}, q_{yz}) = \cap\{p_{xz} \in Q^n_{xz} \mid q^+_{xy} \cap q^+_{yz} \subseteq p^+_{xz}\}$, it follows that $res(q_{xy}, q_{yz}) \in Q^n_{xz}$.

Variable elimination is then defined by updating each two variable projection with resolvants and removing all two variable projections over the variable to be eliminated. That is, where $[\![S]\!] = q_X$, $\exists y.q_X = [\![\{q_{uv} \in S \mid y \notin \{u, v\}\} \cup \{q_{xz} \cap res(q_{xy}, q_{yz}) \mid q_{xy}, q_{yz}, q_{xz} \in S\}]\!]$.

4.4 Completion

Completion is the operation in weakly relational domains through which the various two dimensional projections in the representation communicate with each other. An element of the quadtree domain is complete if no two variable component can be tightened whilst leaving the higher dimensional quadtree unchanged. Formally, let $q_X = [\![S]\!]$. S is complete if whenever $q_X = [\![S']\!]$ and $q_{xy} \in S$, $q'_{xy} \in S'$ then $q_{xy} \subseteq q'_{xy}$.

Completion can be computed by recursively updating a representation S by S'. If $q_{xy}, q_{yz} \in S$, then $S' = (S \setminus \{q_{xz}\}) \cup \{q_{xz} \cap res(q_{xy}, q_{yz})\}$. This rule is applied until any selection of q_{xy}, q_{yz} results in $S' = S$. Termination is ensured as the Q^n_X lattice is finite.

Completion is a crucial component of a number of domain operations as specified in this section. The application of meet does not require completion and variable elimination can be thought of as partial completion, whereas join and entailment require the representation to be complete. However, it will be argued in section 5 that with boolean representations of quadtrees completion is an unnecessary operation.

4.5 Join

Suppose that $[\![S_q]\!] = q_X \in Q^n_X$, $[\![S_p]\!] = p_X \in Q^n_X$ and that S_p, S_q are complete. Then join can be determined pairwise on the individual quadtree components. Over a variable pair, join is simply union: define $q_{xy} \sqcup p_{xy} = q_{xy} \cup p_{xy}$. This lifts to domain elements: $q_X \sqcup p_X = [\![\{q_{xy} \sqcup p_{xy} | x, y \in X, q_{xy} \in S_q, p_{xy} \in S_p\}]\!]$.

4.6 Entailment

Entailment can be determined in terms of pairwise entailment on the individual quadtree components, but again completion is required. Suppose that $[\![S_q]\!] = q_X \in Q^n_X$, $[\![S_p]\!] = p_X \in Q^n_X$ and that S_q is complete. Over a variable pair, entailment is containment: $q_{xy} \models p_{xy}$ if and only $q_{xy} \subseteq p_{xy}$. This lifts to domain elements: $q_X \models p_X$ if and only if $q_{xy} \models p_{xy}$ for each $\{x, y\} \in X$, $q_{xy} \in S_q$, $p_{xy} \in S_p$.

4.7 Abstraction

The abstraction of a set $R \subseteq I^d$ is given by $\alpha(R) = \sqcap\{q_X \in Q^n_X | R \subseteq q_X\}$. Concretisation is simply the identity. The weakly relational nature of quadtrees induces a loss of information for three (and higher) dimensional regions, as the following example illustrates. Suppose that $R = (I \times I \times \{min\}) \cup (I \times \{min\} \times I) \cup (\{min\} \times I \times I)$. Then $\alpha(R) = I \times I \times I$.

Abstraction is potentially expensive. Consider for example, a chessboard of $2^n \times 2^n$ squares where the dark square are the region of interest. The quadtree describing this has maximum size, that is $(4^{n+1} - 1)/3$ nodes. This is potentially problematic. However, this problem might be addressed in at least two ways. One approach is to restrict the granularity to throttle the size of the representation. The other, complementary, approach is to table commonly occurring programming constructs, allowing abstraction via lookup.

Assignment can be handled as in the TVPI domain [21] by introducing a fresh variable. Consider, for example, the assignment $x := x+1$. This becomes $x' = x + 1$, which is abstracted and the meet of the result with the current domain element is calculated. x is then projected out and x' is renamed to x. Note the importance of variable elimination to this approach.

4.8 Widening

Note that Q^I_X forms a finite lattice, hence widening is not strictly necessary to enforce termination, even if fixpoint acceleration is desirable.

As noted by [13] the choice of widening is key to getting a domain to perform well in an analysis. One widening for quadtrees naturally suggests itself: increase the granularity as the number of iterations increases. Formally this is as follows (and is parameterised by a function associating an iterate with a granularity).

Suppose that $[\![S_q]\!] = q_X \in Q^n_X$, $[\![S_p]\!] = p_X \in Q^n_X$, where q_X represents the j^{th} iterate of analysis and p_X the $(j+1)^{\text{th}}$ iterate. The expectation is that S_p, S_q are complete, although this is not strictly necessary. Over a variable pair, widening is as follows: $q_{xy} \nabla p_{xy} = q_{xy} \cup (\cap\{r_{xy} \mid r_{xy} \in Q^{n,k}_{xy}, p_{xy} \setminus q_{xy} \subseteq r_{xy}\})$, where the granularity k is a history dependent value. This then lifts to domain elements: $q_X \nabla p_X = [\![\{q_{xy} \nabla p_{xy} \mid x, y \in X, q_{xy} \in S_q, p_{xy} \in S_p\}]\!]$.

Further discussion of widening quadtrees represented as booleans, or rather not doing so, is given in section 5.

5 Boolean Formulae for Quadtrees

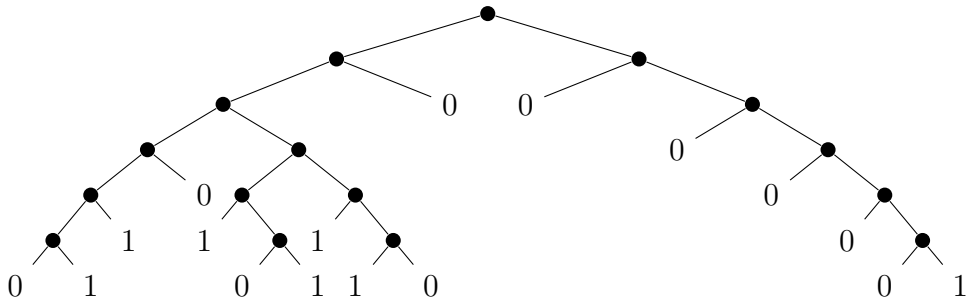
Elements of the quadtrees domain can be represented easily by structures for boolean formulae. This section details the encoding of quadtrees into Binary Decision Diagrams (BDDs) and formulae in conjunctive normal form (cnf), as well as discussing the implications of these encodings.

A quadtree over x and y, Q^n_{xy}, has associated with it $2n$ variables. That is, one variable for each dimension and each permitted square size. These variables will be referred to by x_i and y_i, where i is the power describing the width of the corresponding squares. It is important to note that when the same axis occurs in different quadtrees, the same boolean variables are used. Satisfying assignments over these variables then correspond to the region of interest described by a quadtree.

5.1 Quadtrees as BDDs

The encoding of a quadtree as a BDD is straightforward. The four children of a node in the quadtree become four leaves of a BDD over two variables. That is, the nodes correspond to (x_i, y_i) pairs as follows: NE to $(1,1)$, NW to $(0,1)$, SW to $(0,0)$, SE to $(1,0)$.

The quadtree in section 2 is represented by the following OBDD (reduction omitted for presentational purposes). The ordering is $[x_2, y_2, x_1, y_1, x_0, y_0]$ and left branches correspond to *true*.



A multi-rooted ROBDD will then describe a quadtree over many dimensions, exploiting structural similarity to obtain a compact representation.

5.2 Quadtrees as cnfs

The counterpart of the reduced disjunctive normal form of BDDs is conjunctive normal form, here not reduced. The clauses can be thought of as each describing a region of the grid not captured by the quadtree. That is, a counter-model to each clause describes a region not in the quadtree.

The following is the cnf describing the quadtree from section 2:

$$
\begin{array}{ll}
\neg x_2 \vee \neg y_2 \vee \neg x_1 \vee \neg y_1 \vee \neg x_0 \vee \neg y_0 & \wedge \\
\neg x_2 \vee \neg y_2 \vee x_1 \vee \neg y_1 \vee x_0 \vee \neg y_0 & \wedge \\
\neg x_2 \vee \neg y_2 \vee x_1 \vee y_1 \vee x_0 \vee y_0 & \wedge \\
\neg x_2 \vee \neg y_2 \vee \neg x_1 \vee y_1 & \wedge \\
x_2 \vee \neg y_2 & \wedge \\
x_2 \vee y_2 \vee \neg x_1 \vee \neg y_1 & \wedge \\
x_2 \vee y_2 \vee x_1 \vee \neg y_1 & \wedge \\
x_2 \vee y_2 \vee x_1 \vee y_1 \vee \neg x_0 \vee \neg y_0 & \wedge \\
x_2 \vee y_2 \vee x_1 \vee y_1 \vee x_0 \vee \neg y_0 & \wedge \\
x_2 \vee y_2 \vee x_1 \vee y_1 \vee \neg x_0 \vee y_0 & \wedge \\
x_2 \vee y_2 \vee \neg x_1 \vee y_1 & \wedge \\
\neg x_2 \vee y_2 &
\end{array}
$$

The various two dimensional projections give rise to cnf formulae and conjoining these gives a single cnf describing the higher dimensional quadtree.

5.3 Avoiding completion

Completion is an apparently crucial operations in weakly relational domains, and the treatment of quadtrees in section 4 is no different. However, notice

that when a quadtree is represented by a boolean formula in cnf completion becomes less compelling. Completion can easily be calculated by resolution steps, but these steps simply add redundent clauses to the representation. All of the domain operations can be performed by their logical equivalents at the level of cnf formulae without applying completion and it is not clear that completion leads to any computational advantage.

The same tactic can be applied with the BDD representation – simply take the conjunction of the two variable projections. However, note that since ROBDDs give a canonical representation this tactic corresponds to calculating the completion since completion aspires to a canonical representation.

In order to reflect on the two representations, consider their complexities. Both ROBDDs and cnf are have potentially exponentially large representations. The following tabulates complexity of the core domain operations (where N is the size of the input):

	$\wedge$	$\vee$	$\exists x$	$\models$	$\equiv$	$\forall x$
ROBDD	$O(N^2)$	$O(N^2)$	$O(N)$	$O(N)$	$O(1)$	$O(N)$
cnf	$O(1)$	$O(N^2)$	$O(N^2)$	$O(2^N)$	$O(2^N)$	$O(1)$

The gain by using cnf comes from the low complexity of conjunction, the result of the non-canonical representation. This is offset by the cost of the entailment/equivalence and projection. Entailment for cnf is implemented by SAT solving and although of high theoretical complexity, SAT solving has been demonstrated to be surprisingly tractable on very large structured problems. Projection is important to this approach since it is required in the treatment of assignment and the relatively high cost of this operation might prove to be prohibitive. The answers to these performance questions are left open, but it is noted that implementation work for dependency analysis has demonstrated that cnf is an attractive representation [14].

5.4 Avoiding widening

As noted above, defining suitable widening operations is one of the most difficult tasks in numeric domains. Recent work [18] on the automatic derivation of transfer functions offers a promising way forward – it shows how least solutions to fixpoint equations can be derived symbolically by applying universal quantifier elimination over systems of linear inequalities. This finesses the need for widening. The tactic amounts to stating that the least solution both constitutes a solution and is smaller than every other solution (hence the need for universal qualification). The domain of quadtrees is ordered by entailment in its boolean encoding which suggests that forall elimination can be applied to directly compute least fixpoints without employing widening. This would

provide a spatial analogue of immediate fixpoint calculation [22], which has been applied to directly compute fixpoints over the domain of positive boolean functions.

6 Related Work

The work contained in this paper can be viewed as a weakly relational domain that is to finite powersets of intervals, as TVPI is to polyhedra. The use of finite powersets of intervals has received some attention recently. In [2] the authors are concerned with widenings for powerset domains in general, whilst in [20] the focus is on how to analyse across paths (something that powerset domains are well suited to) whilst retaining the more attractive computational properties belonging to the base domains owing to their path summarisation.

However, the closest work to that presented here is that of Gurfinkel and Chaki. In [4] LDDs are introduced. These are BDD like structures where nodes are interpreted as linear inequalities, giving a decompostion of n-dimensional space into (a finite number of) regions of interest. In [13] a domain that corresponds to finite powersets of intervals is given. The domain is represented as LDDs (in fact, a restriction of LDDs, since only single variable inequalities are required) leading to an attractive analysis that appears to scale.

A completely different approach to representing non-convex spaces is to use congruences as discussed in [1].

In the context of bounded model checking quadtrees have been used to model electrical fields [12] as part of a system to diagnose spirals of electrical activity indicating cardiac abnormalities. Note that spirals are particularly suited to the widening presented in section 4.8. Finally, the paper uses quadtrees with back arcs in order to define fractals, which suggests that quadtrees can have dense representation akin to BDDs with back arcs.

7 Discussion and future work

Quadtrees have a vast literature. They have been generalised to higher dimension (octrees) and applied in diverse applications. They do not necessarily have to represent a square grid structure which offers another degree of expressive freedom. Exploiting the quadtree literature is one avenue of future work. In tandem with this existing implementations of quadtrees will be investigated for their suitability in program analysis. Theoretical questions such as optimality of the operators presented remain open, and such questions will be subject to further enquiry.

In [7] it is noted that successful analyses result, in part, from careful selection of component domains. This motivates research into new domains that might earn their place in the toolkit. It is not yet clear how effective quadtrees

will be for program analysis, therefore the advantages and disadvantages of quadtrees are given by way of summary. Advantages include:

- they are a weakly relational domain not based on inequalities
- they can describe spaces that are not necessarily convex or linear
- they can be encoded in propositional logic, allowing use of BDDs and SAT
- they come with a natural form of widening, though their propositional link suggests that widening may not be required at all
- the granularity can be throttled to control the size of the representation
- the technique does not inherit the problem of storing and manipulating large coefficients that often arise with linear inequalities [5].

Disadvantages include:

- the data-structure is potentially large
- it is not clear how to effectively deal with abstraction
- the proposed widening is natural, but it is also crude and it is not clear how much information will be preserved.

Acknowledgements The authors would like to thank Sagar Chaki and Arie Gurfinkel for making [13] available ahead of publication. They would also like to thank Karl Newman-Smart for discussion and his help with the diagrams. Finally they would like to thank the anonymous referees for helpful comments.

References

[1] R. Bagnara, K. Dobson, P. M. Hill, M. Mundell, and E. Zaffanella. Grids: A Domain for Analyzing the Distribution of Numerical Values. In *Logic-Based Program Synthesis and Transformation*, volume 4407 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2006.

[2] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening Operators for Powerset Domains. *International Journal on Software Tools for Technology Transfer*, 8(4-5):449–466, 2006.

[3] R. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[4] S. Chaki, A. Gurfinkel, and O. Strichman. Decision Diagrams for Linear Arithmetic. In *International Conference on Formal Methods in Computer-Aided Design*, pages 53–60. IEEE Press, 2009.

[5] P. J. Charles, J. M. Howe, and A. King. Integer Polyhedra for Program Analysis. In *Algorithmic Aspects in Information and Management*, volume 5564 of *Lecture Notes in Computer Science*, pages 85–99. Springer, 2009.

[6] R. Clarisó and J. Cortadella. The Octahedron Abstract Domain. *Science of Computer Programming*, 64:115–139, 2007.

[7] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Why does Astrée Scale Up? *Formal Methods in System Design*, 35(3):229–264, 2009.

[8] P. Cousot, R. Cousot, and L. Mauborgne. A Scalable Segmented Decision Tree Abstract Domain. In Z. Manna and D. Peled, editors, *Pnueli Festschrift*, volume 6200 of *Lecture Notes in Computer Science*, pages 72–95. Springer, 2010.

[9] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *Principles of Programming Languages*, pages 84–97. ACM Press, 1978.

[10] M. de Berg, O. Cheong, M. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[11] R. A. Finkel and J. L. Bentley. Quad Trees: a Data Structure for Retrieval on Composite Keys. *Acta Informaticae*, 4:1–9, 1974.

[12] R. Grosu, S. A. Smolka, F. Corradini, A. Wasilewska, E. Entcheva, and E. Bartocci. Learning and Detecting Emergent Behavior in Networks of Cardiac Myocytes. *Communications of the ACM*, 52(3):97–105, 2009.

[13] A. Gurfinkel and S. Chaki. BOXES: A Symbolic Abstract Domain of Boxes. In *Static Analysis Symposium*, Lecture Notes in Computer Science. Springer, 2010. Forthcoming.

[14] J. M. Howe and A. King. Positive Boolean Functions as Mulitheaded Clauses. In *International Conference on Logic Programming*, volume 2237 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2001.

[15] J. M. Howe and A. King. Logahedra: a New Weakly Relational Domain. In *Automated Technology for Verification and Analysis*, volume 5799 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2009.

[16] F. Logozzo and M. Fähndrich. Pentagons: a Weakly Relational Abstract Domain for the Efficient Validation of Array Accesses. In *ACM Symposium on Applied Computing*, pages 184–188. ACM Press, 2008.

[17] A. Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[18] D. Monniaux. Automatic Modular Abstractions for Linear Constraints. In *Principles of Programming Languages*, pages 140–151. ACM Press, 2009.

[19] M. Müller-Olm and H. Seidl. Analysis of Modular Arithmetic. *Transactions on Programming Languages and Systems*, 29(5), 2007.

[20] S. Sankaranarayanan, F. Ivanic, I. Shlyakhter, and A. Gupta. Static Analysis in Disjunctive Numerical Domains. In *Static Analysis Symposium*, volume 4134 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2006.

[21] A. Simon, A. King, and J. M. Howe. Two Variables per Linear Inequality as an Abstract Domain. In M. Leuschel, editor, *Logic Based Program Development and Transformation*, volume 2664 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2002.

[22] H. Søndergaard. Immediate Fixpoints and Their Use in Groundness Analysis. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 1996.