Improved Precision in Polyhedral Analysis with Wrapping

Stefan Bygde,[1] Björn Lisper,[1] Niklas Holsti[2]

[1] School of Innovation, Design, and Engineering, Mälardalen University, SE-721 23 Västerås, Sweden
{stefan.bygde,bjorn.lisper}@mdh.se
[2] Tidorum Ltd, Helsinki, Finland
niklas.holsti@tidorum.fi

Abstract. Abstract interpretation using convex polyhedra is a common and powerful program analysis technique to discover linear relationships among variables in a program. However, the classical way of performing polyhedral analysis does not model the fact that values typically are stored as fixed-size binary strings and usually have wrap-around semantics in the case of overflows. In resource-constrained embedded systems, where 8- or 16-bit processors are used, wrapping behaviour may even be used intentionally to save instructions and execution time. Thus, to analyse such systems accurately and correctly, the wrapping has to be modelled.

We present an approach to polyhedral analysis which derives polyhedra that are bounded in all dimensions. Our approach is based on a previously suggested wrapping technique by Simon and King [19], combined with limited widening, a suitable placement of widening points and size-induced restrictions on unbounded variables. With this method, we can derive fully bounded polyhedra in every step of the analysis.

We have implemented our method and Simon and King's method compared them. Our experiments show that for a suite of benchmark programs it gives at least as precise result as Simon and King's method. In some cases we obtain a significantly improved result.

Keywords: Abstract Interpretation, Abstract Domains, Numerical Domains, Convex Polyhedra, Widening, Overflows, Wrapping

1 Introduction

A common program analysis is *value analysis*, which finds safe approximations of the possible numerical values the program variables can take at each point in a program. This is typically done using abstract interpretation [6] with some numerical abstract domain. Many relational and non-relational such domains have been developed [6, 8, 9, 16, 7], with the common assumption that variables can take arbitrarily large integer values. However, in real programs, a value is usually stored as a fixed-size binary string. This introduces the risk of *overflows*, meaning that a value is too large to be stored in a binary string of the given size. An overflow could result in a run-time error, saturation of the result at the largest or smallest representable value of the type, or, for integers, a wrap-around. It is not uncommon that wrap-arounds are used intentionally, in particular on resource-constrained processors with a short word-length, to save instructions and clock cycles. Thus, to analyse such programs it is crucial to have abstract numerical domains that are sound also in the presence of wrap-arounds. The goal of our work is to advance the state of the art regarding such domains, and their use in value analysis, to widen the scope of such analyses to software for small embedded systems.

1.1 Related Work

Sen and Srikant [18] present a variation of the reduced product of the integer and congruence domain which handles special cases when overflow occurs. In addition they use a relational analysis of affine equalities. Gustafsson et al. [11] modify the interval domain so that variables are bounded to within their size-induced range, and wrap-arounds are handled by using a more powerful representation of intervals. Relational domains are more challenging. Müller-Olm and Seidl [17] present an analysis that can derive all affine equalities (but not inequalities) among variables of programs which is safe in the case of wrap-arounds. Brauer and King [3] suggest a method to derive transfer functions for relational domains, and do so for the octagon domain, while considering wrap-around effects by using a SAT-solver. Finally, Simon and King [19] present a way to use classical polyhedral analysis [7] soundly for programs with wrap-around semantics for integers. Our work builds on this approach.

1.2 Contributions

We present a polyhedral analysis, originally introduced in [5], which is sound in the presence of wrap-arounds. Our approach is based on a combination of wrapping polyhedra (using the approach in [19]), limited widening [14], an appropriate placement of the widening, and imposing bounds on variables based on type information.

Our original motivation for developing this analysis is the use of polyhedral value analyses in the parametric loop bounds analysis described in [15, 4]. This loop bounds analysis requires a relational abstract domain, and the results are typically used in a subsequent parametric Worst-Case Execution Time (WCET) analysis [21]. Thus, to make such analysis sound for software using wrap-arounds, a sound value analysis is required. As mentioned wrap-arounds are not uncommon in code for small embedded processors, which are commonly used in embedded time-critical applications. However, our results are general and by no means restricted to this particular application.

The major benefit of our approach is the increased precision compared to using the approach outlined in [19], while retaining the soundness in the presence of wrap-arounds.

Section 2 contains preliminaries to our approach, including terminology and an explanation of Simon & King's method. In Section 3 we describe our bounded polyhedral analysis. Section 4 describes the implementations, and an evaluation and comparison of the methods is given in Section 5. We conclude with a summary of the results in Section 6.

2 Preliminaries

2.1 The Polyhedral Domain

The classical abstract domain of convex polyhedra [7] is a powerful relational abstract domain that can capture linear inequalities among program variables. We will call this domain CP and define it as follows. Let V be the set of numerical variables in a program with $|V| = n$. A linear inequality (or simply *constraint*) c is of the syntactic form

$$c = \sum_{i=0}^{n-1} a_i v_i \leq k$$

where $a_i, k \in \mathbb{R}$ and $v_i \in V$. We then define a convex polyhedron as a finite set of linear inequalities P. Using this definition means that a polyhedron is synonymous with a set. The universal polyhedron (top), then is $P = \top = \varnothing$, i.e., the polyhedron without constraints, whereas the empty polyhedron (bottom) is represented by an inconsistent constraint such as $P = \bot = \{0 \leq -1\}$.

A convex polyhedron can be interpreted as a set of points in the concrete domain of values in $\mathbb{R}^n$ space via the concretisation function:

$$\gamma_{\mathrm{CP}}(P) = \{\mathbf{x} \in \mathbb{R}^n | \forall c \in P : c \vdash \mathbf{x}\}$$

where $c \vdash \mathbf{x}$ means that the point $\mathbf{x}$ fulfills the constraint c, or formally:

$$\sum_{i=0}^{n-1} a_i v_i \leq k \vdash \mathbf{x} \Leftrightarrow \sum_{i=0}^{n-1} a_i x_i \leq k$$

We will also use $P \sqsubseteq Q$ to mean that P is a subpolyhedron of Q, that is, the conjunction of the constraints in P implies each of the constraints in Q, or equivalently, that $\gamma_{\mathrm{CP}}(P) \subseteq \gamma_{\mathrm{CP}}(Q)$.

The methods we investigate in this paper are only concerned with integers, so the term *variable* will refer to an integer-variable. Furthermore, we restrict the image of γ_{CP} to $\mathbb{Z}^n$ by simply ignoring all non-integers in the result.

2.2 Simon and King's method

In [19], Simon and King presented a method to make classical polyhedral analysis sound for programs with wrapping integer semantics. Since our method is directly based on theirs, we summarise their method here, but refer to the original publication for details. To distinguish their method from others, we will from now on refer to it as SK.

Definitions. SK assumes that each variable has a size and a type; the size is the number of bits used to store the integer and the type is whether the value of the variable is to be interpreted as signed or unsigned. From this information we can associate a set of constraints for each variable. We define $R(v) = \{l \leq v, v \leq u\}$ to be the set of *range constraints* for v dictated by the size and interpretation of v. For example, if v_0 is an 8-bit signed variable, then $R(v_0) = \{-128 \leq v_0, v_0 \leq 127\}$. We define the set of range constraints for a set of variables $S \subseteq V$ as

$$R(S) = \bigcup_{v \in S} R(v)$$

$R(V)$, which is the set of range constraints for all the variables, is called the *base window.* This concept is important because it represents an invariant of the concrete environment at any point of the execution of a program.

Next we define a specific modulo operator on integers. Let x be an integer, and v be a variable then we define

$$x \mod v = l + (x \mod (u - l))$$

where l, u comes from $R(v) = \{l \leq v, v \leq u\}$. For example, if $R(v_0) = \{-128 \leq v_0, v_0 \leq 127\}$, then $129 \mod R(v_0) = -128 + (129 \mod 255) = 1$. As can be seen, this modulo operation simulates the wrap-around effect when value x is stored in variable v. This operation can lifted to a point $\mathbf{x} \in \mathbb{Z}^n$ modulo a set of variables S:

$$\langle x_0, ..., x_{n-1} \rangle \mod S = \langle x_0 \mod s_0, ..., x_{n-1} \mod s_{n-1} \rangle$$

where

$$s_i = \begin{cases} v_i \text{ if } v_i \in S \\ \infty \text{ if } v_i \notin S \end{cases}$$

where $x \mod \infty = x$.

Implicit wrapping. SK defines the concretisation function γ_{SK} as

$$\gamma_{\text{SK}}(P) = \{\mathbf{x} \mod V | \mathbf{x} \in \gamma_{\text{CP}}(P)\}$$

This interpretation does not affect the actual analysis in any way, but correctly interprets all values to be within the base window. Note that the points in $\gamma_{\text{SK}}(P)$ will not necessarily form a convex polyhedron. This implicit wrapping works for the analysis of equality-based linear transfer functions, such as assignments of linear expressions to variables, because linear transformations commute with modulo. However, polyhedral abstract interpretation also includes adding linear *constraints* to polyhedra, which typically happens at linear conditionals. Adding a linear inequality constraint does not commute with the modulo operation and therefore Simon and King introduced an explicit wrapping procedure which has to be performed when adding a linear inequality constraint.

Explicit Wrapping. The idea with the explicit wrapping is to make sure that the dimensions of the polyhedron that corresponds to the variables involved in the linear constraint are within their size-induced range before the constraint is added.

Let P be a polyhedron and let σ be a linear constraint involving the variables S_σ. Intuitively, the wrapping procedure consist of first partitioning the subspace of $\mathbb{Z}^n$ involving the variables S_σ into a finite grid of windows of the same size and dimension as $R(S_\sigma)$ as shown in Figure 1(a). If any variable $v \in S_\sigma$ is unbounded in P, then that variable is eliminated from P (see the procedure for this in, for example [7]) and then assumed to only be constrained by $R(v)$, making it possible to create a finite grid.

Then each partition, which is the size of the base window, is shifted to the position of the base window and intersected by σ. Finally, the convex hull of all these shifted and intersected partitions is taken as the result (see Figure 1(b)). The procedure is explained in detail in [19]. The result of wrapping P with constraint σ and a set of variables S is denoted as follows:

$$\texttt{wrap}(P, \sigma, S)$$

It is important to notice that explicit wrapping potentially loses precision of P. This is for two reasons: computing the convex hull is naturally approximate and the elimination of unbounded variables in the condition may lose relational information. Thus, explicit wrapping should be applied only when necessary.

3 Bounded Polyhedral Analysis

In this section we present our method of bounded polyhedral analysis. Our idea was first presented in [5]. Our approach is an extension of SK: we use γ_{SK} as the concretisation function, and we do explicit wrapping at conditionals.

The main idea of our approach is to reduce the imprecision introduced in SK by the wrapping. As explained, wrapping may introduce loss of precision, especially for unbounded polyhedra. Thus, our approach is to make an analysis where unbounded polyhedra never occur. An illustrating example of the approach can be found in [5]. From now on, we will refer to our approach as BD.

3.1 Unboundedness in Polyhedral Analysis

When performing abstract interpretation on a program, a polyhedron may become unbounded in three cases: First, at the initial program point, where nothing is known about the program variables. Second, any non-linear assignment will eliminate the assigned variable from the constraints of the polyhedron, leaving it unbounded in the dimension the assigned variable represent. Third, classical widening often produces an unbounded polyhedron since it removes constraints.

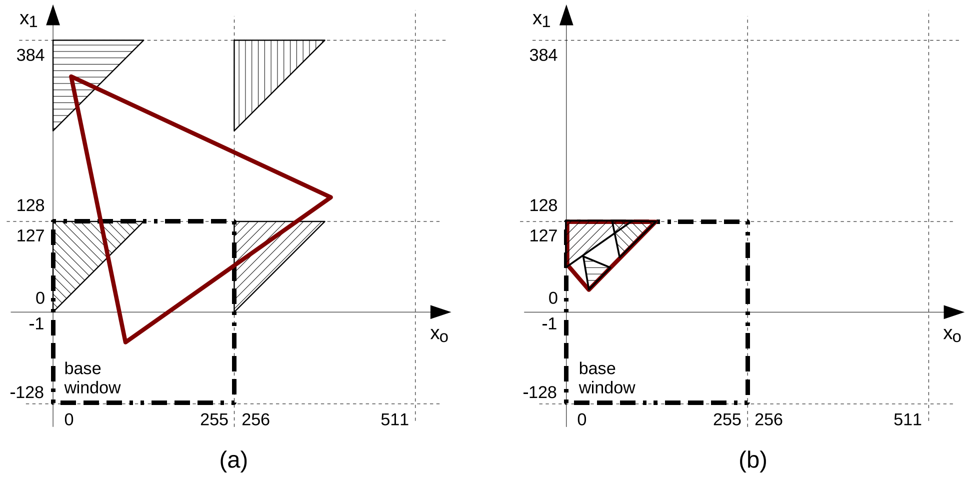

Fig. 1. The picture on the left (a), shows a polyhedron before wrapping. The base window is shown outlined by a dot-dashed square. The polyhedron covers a part of the base window and parts of the three neighbouring windows. The grid of variously hatched triangles shows the condition $x_0 \leq x_1$ taken as a signed comparison of the 8-bit unsigned residue of x_0 with the 8-bit signed residue of x_1. The polyhedron intersects three components of this condition, one in the base window, one in the next window to the right of the base window, and one in the next window above the base window. To the right (b), the intersections of the condition with the unwrapped polyhedron are shown, shifted to the base window, and their convex hull, which is the resulting wrapped polyhedron.

3.2 Making Polyhedra Bounded

We consider each of the possible ways of making a polyhedron unbounded and show how to soundly and precisely make it bounded.

One way of defining the abstract semantics of a program is through a function τ taking a program point (typically an edge in the flow chart or CFG of a program) and returning an abstract environment, in this case a convex polyhedron. The function is typically defined as a recurrence equation, $\tau_0(p) = \bot$ for all program points p, and $\tau_n(p)$, where $n > 0$, is defined in terms of τ_{n-1}. Fixed point iteration is then used to find a solution to these equations.

The definition of this function τ^{CP} for classic polyhedral abstract interpretation can be found in [7]. In this section we will define τ^{BD} where it differs from τ^{CP}. We will also note the single case where τ^{SK} differs from τ^{CP}.

Entry point In abstract interpretation, a common assumption is that nothing is known about the values of program variables at the entry point of the program. In CP, this is represented by the polyhedron with no constraints. Let p be the entry point of the program. Then:

$$\tau_n^{\mathrm{CP}}(p) = \varnothing$$

However, in SK and BD, each integer variable is associated with a type and a size. As explained in Section 2.2, the set of possible points lies within the base window. Consequently, in BD we assign the whole base window as the polyhedron at the entry point of the program.

$$\tau_n^{\mathrm{BD}}(p) = R(V)$$

This is sound and more precise than CP and SK.

Non-Linear Assignments In CP and SK, a non-linear assignment discards all information about the assigned variable by eliminating it from the set of constraints. Let p_0 represent the program point before the non-linear assignment `x := ?`, and let p_1 be the program point directly after. Then,

$$\tau_n^{\mathrm{CP}}(p_1) = \exists_x(\tau_n^{\mathrm{CP}}(p_0))$$

where $\exists_x(P)$ represents the polyhedron P where the variable x has been eliminated. Since x is not involved in any constraint after elimination, it is safe to assume that it is within its range.

$$\tau_n^{\mathrm{BD}}(p_1) = \exists_x(\tau_n^{\mathrm{CP}}(p_0)) \cup R(x)$$

Again, this is sound and more precise than SK and BD.

Widening For a program whose control-flow graph (CFG) contains cycles, widening is necessary to ensure termination of the analysis. In classical abstract interpretation, the widening is usually inserted immediately where the program flow joins in a cycle. The classical widening operation removes unstable bounds from the polyhedron which often results in an unbounded polyhedron. However, removing single constraints does not necessarily destroy all relational information among variables, so it would *not* be safe to apply any range constraints after doing widening. We have to use another approach.

3.3 Making Widening Bounded

The standard widening operation, as mentioned, often makes polyhedra unbounded. However, with the help of *limited widening* it might be possible to intersect the result with a finite number of constraints. Our idea is to use widening in such a way that it is always possible to intersect the result with a fully bounded polyhedron.

Limited Widening Limited widening was suggested in [14]. The idea with limited widening is to have a set of constraints C and define limited widening ∇_C as follows:

$$P \nabla_C Q = (P \nabla Q) \cup \{c \in C | P \sqsubseteq \{c\} \wedge Q \sqsubseteq \{c\}\}$$

That is, the result of the widening is intersected with all constraints in C which hold in both P and Q. It can be shown that this is a widening operation for any set of constraints C. The set C is typically selected strategically for each program.

Our idea is to use a limited widening such that $C \sqsubseteq R(V)$. Our goal is to be able to intersect the result of the widening with at least $R(V)$, to make the polyhedron fully bounded.

Placement of the Widening Points Since our goal is to safely intersect the polyhedron with the base window, we can see from the definition of limited widening that both arguments to the widening operator have to be fully contained within the base window. Fortunately, there are certain points in the program where this is always true: at explicit wrappings, i.e., at conditionals.

SK requires that polyhedra are explicitly wrapped when conditionals are applied. Wrapping guarantees that $\texttt{wrap}(P, \sigma, S) \sqsubseteq R(S)$, thus if $S = V$ the result will be fully contained in the base window. For this reason, our strategy is to make sure widening is done in conjunction with wrappings. To avoid unnecessary loss of precision, the widening points cab be distributed so that each cycle in the CFG contains exactly one widening point.

Specifically, this means that widening is applied where the control flow takes a conditional jump due to evaluating the constraint σ to true (if program flow enters a cycle from evaluating a conditional to false, it can still be seen as evaluating its negation to true). The exact placement of widening points in the cycles is not dictated by BD, as long as they are placed at conditionals.

Let p_0 be the program point before a conditional, and let p_1 represent the point where control flow enters a cycle from evaluating the condition σ to true. If σ is non-linear, σ is considered to be always be true. Classical polyhedral analysis just adds the condition as a new constraint to the polyhedron (widening is typically done elsewhere):

$$\tau_n^{\mathrm{CP}}(p_1) = \tau_n^{\mathrm{CP}}(p_0) \cup \{\sigma\}$$

In SK, the polyhedron is wrapped,

$$\tau_n^{\mathrm{SK}}(p_1) = \texttt{wrap}(\tau_n^{\mathrm{SK}}(p_0), \sigma, S_\sigma)$$

In BD, this is where widening is done, in particular limited widening:

$$\tau_n^{\mathrm{BD}}(p_1) = \tau_{n-1}^{\mathrm{BD}}(p_1) \nabla_{R(V) \cup \{\sigma\}} (\texttt{wrap}(\tau_n^{\mathrm{BD}}(p_0), \sigma, V))$$

Note that all variables V will be wrapped to ensure that the resulting polyhedron is contained within the base window: this differs from SK, where only the variables that appear in the condition are wrapped. This will potentially be less precise than wrapping with S_σ, but it will always result in a fully bounded polyhedron, stated formally as:

Theorem 1. *Let p be a program point where control flow enters a cycle from evaluating the condition σ to true. Then,*

$$\tau_n^{\mathrm{BD}}(p) \sqsubseteq R(V)$$

for all $n \geq 0$.

A proof can be found in [5].

4 Implementation

In order to do an evaluation of bounded polyhedra we have implemented both SK and BD in the static analysis tool SWEET [1, 13].

4.1 SWEET and ALF

SWEET is a Worst-Case Execution Time (WCET) analysis tool, but it uses a number of general program analysis techniques, including value analyses based on abstract interpretation, and it can be used as a pure value analysis tool. SWEET analyses code on an intermediate format called ALF [12]. ALF is a language designed specifically for program analysis, and to be able to represent both source and object code faithfully. Currently, translators exist from C to ALF and from PowerPC binaries to ALF. In our evaluation, we have analysed C code translated into ALF.

4.2 Analysis Details

We implemented polyhedral analysis in SWEET using the Parma Polyhedra Library [2]. Conveniently, this library has an implementation of the explicit wrapping procedure presented in [19].

Our current implementations of BD and SK do not distinguish individual fields of structs, and individual elements of arrays. Currently we use the safe approximation to give array elements and elements of structs the top value in the analysis.

In the evaluation, we have applied widening in loops at the exits of their header basic blocks that do not exit the loop. This ensures that each path in the loop has exactly one widening point. All of the programs that we have analysed in our evaluation have structured loops (meaning that each loop has exactly one header), so it is possible to place widening in this way. Other placements of widening points are also possible, but this has not been investigated.

5 Evaluation

Our aim is to compare the precision of the polyhedra resulting when analysing programs with SK and BD. Our evaluation consists of analysing a set of benchmark programs with both SK and BD. Abstract interpretation yields results for all points in the analysed program, in our case each basic block is associated with a polyhedron. For each of basic block p of a program, we have a polyhedron SK(p) resulting from analysis with SK and a polyhedron BD(p) resulting from analysis with BD. While the usefulness of the results should ideally be compared considering a particular application, we have settled on two ways of comparison. The first one is to investigate the relation between the sets $\gamma_{\mathrm{SK}}(SK(p))$ and $\gamma_{\mathrm{SK}}(BD(p))$ in terms of inclusion. However, this does not show how much better the result is, just if it is better or not. For this reason we also measure the precision of a result by the number of integer points it contains. This measurement is highly relevant in for instance the WCET analysis in [15, 4], which is directly based on counting the number of integer points inside polyhedra derived from abstract interpretation.

5.1 Computing the set of integers of a polyhedron

We concluded in Section 2 that $\gamma_{\mathrm{SK}}(P)$ does not necessarily form a convex polyhedron. This means we cannot use conventional techniques to compare and count elements in convex polyhedra when dealing with these kind of sets. However, if P is a fully bounded polyhedron, then $\gamma_{\mathrm{SK}}(P)$ is the union of a finite set of convex polyhedra within the base window. The Barvinok library [20] provides techniques for manipulating and counting sets of integers, including unions of convex polyhedra. Thus, as long as the polyhedra are fully bounded, we can use Barvinok to compare them inclusion-wise as well as count the size of these sets.

If P is unbounded, we can no longer see $\gamma_{\mathrm{SK}}(P)$ as a finite union of convex polyhedra and we have to use another technique. We can form the polyhedron $\texttt{wrap}(P, \varnothing, V) \subseteq R(V)$ which is fully contained within the base window. Moreover, $\gamma_{\mathrm{SK}}(\texttt{wrap}(P, \varnothing, V))$ always forms a convex polyhedron, which means that straightforward counting techniques for convex polyhedra can be used. However, $\texttt{wrap}(P, \varnothing, V)$ is potentially less precise than P. Furthermore, both SK and BD are designed to avoid explicit wrapping as much as possible to avoid imprecision. Still, to give an indication of the improvement of BD compared to SK in the cases where SK results in unbounded polyhedra, we use this comparison.

5.2 The Setup

We used six benchmarks from the Mälardalen Benchmark suite [10]. The programs were translated into ALF using a C-to-ALF compiler and were analysed using SWEET compiled for Windows XP using Cygwin. The experiments were performed on a 2.4 GHz dual core Intel with 3.45 GB ram.

We analysed the benchmarks programs with SK and BD. Each basic block p in the analysed program is associated with a SK polyhedron $SK(p)$ and a BD polyhedron $BD(p)$. In two of the benchmarks, namely *bs* and *bsort100*, $SK(p)$ is unbounded for some basic blocks p. For these two benchmarks we compared the results using $\texttt{wrap}(SK(p), \varnothing, V)$, as described Section 5.1. This is also indicated in the tables.

5.3 The Results

Table 1. Subset Relations

Benchmark	BBs	$\gamma_{\mathrm{SK}}(\mathrm{BD}) \subseteq \gamma_{\mathrm{SK}}(\mathrm{SK})$	$\gamma_{\mathrm{SK}}(\mathrm{BD}) \subset \gamma_{\mathrm{SK}}(\mathrm{SK})$
bs (wrap)	11	100%	27%
bsort100 (wrap)	20	100%	40%
fibcall	6	100%	33%
insertsort	7	100%	43%
jcomplex	14	100%	57%
loop3	392	100%	27%
Total	450	100%	29%

Set Relations. Table 1 shows the percentage of program points p in which the set $\gamma_{\mathrm{SK}}(\mathrm{BD}(p))$ is a subset respective proper subset to $\gamma_{\mathrm{SK}}(\mathrm{SK}(p))$.

The column *BBs* shows how many basic blocks the benchmark has, and consequently, how many pairs of polyhedra are compared. As can be seen, BD is in all observed cases at least as precise as SK in terms of inclusion, often strictly better.

Number of Integer Points. Table 2 shows statistics when comparing $|\gamma_{\mathrm{SK}}(\mathrm{BD}(p))|$ to $|\gamma_{\mathrm{SK}}(\mathrm{SK}(p))|$, except for bs and bsort where $|\gamma_{\mathrm{SK}}(\texttt{wrap}(\mathrm{BD}(p)), \varnothing, V)|$ is compared to $|\gamma_{\mathrm{SK}}(\texttt{wrap}(\mathrm{SK}(p)), \varnothing, V)|$. We explain the table below:

Table 2. Comparing number of integer points

Benchmark	BBs	Best	Avg str.	Avg imp.
bs (wrap)	11	< 1%	< 1%	< 1%
bsort100 (wrap)	20	25%	15%	6%
fibcall	6	75%	75%	25%
insertsort	7	< 1%	< 1%	< 1%
jcomplex	14	75%	31%	17%
loop3	392	25%	3%	1%

BBs The number of basic blocks in the benchmark.

Best This column shows the highest percentage-decrease in number of integer points when comparing BD to SK observed in this benchmark.

Avg str. This shows the average decrease in number of integer points when comparing BD to SK, in the cases where there is a strict improvement. The percentage of polyhedra with strict improvement can be seen in Table 1.

Avg imp. This shows the average decrease in number of integer points inside polyhedra when comparing BD to SK, including cases where there was no strict improvement.

In total 450 SK polyhedra have been compared to 450 BD polyhedra. As seen in Table 1, BD is at least as good as SK in all cases, and in 29% of the total cases strictly better. The amount of improvement varies from less than one percent, to 75% in some polyhedra. As we will see in the following section, this improvement comes with little cost.

5.4 On Efficiency

To be able to compare the precision of the analyses, the SK and BD methods were run simultaneously in the above experiments. To also give an impression of the complexity of the two approaches, we have run the analyses in isolation on the two biggest benchmarks (in terms of program points): *loop3* and *bsort100*. We compare them with two criteria: the running times and the number of iterations before termination. Note that the analysis implementastion has not been optimised and contains a lot of run-time checks and debugging information. The running times are just intended to give a rough idea of the difference in performance. The running times were acquired by the single user real-time component of the result of the Cygwin command `time`.

Table 3 compares the running times and iterations of three methods, the classical polyhedral analysis (CP), Simon and King's method (SK) and our bounded polyhedral method (BD).

Table 3. Complexity

Benchmark	Running Time			Iterations		
	CP	SK	BD	CP	SK	BD
loop3	1m 2.81s	1m 14.84s	1m 24.62s	878	1054	1082
bsort100	4.64s	4.98s	5.33s	34	35	48

As can be seen, BD takes a few more iterations to terminate than SK due to the more complex polyhedra resulting. This also shows that SK takes more iterations than classical polyhedral analysis. It can be concluded that the increased cost of running BD compared to SK is quite limited.

6 Summary and Conclusions

We have developed an analysis using fully bounded convex polyhedra which is sound for programs with wrap-around semantics. The method is based on earlier work by Simon & King [19] but has some additional features to compensate for imprecision. This is done by imposing range bounds on variables at the initial program point and at non-linear assignments, wrapping polyhedra at conditionals and finally by using limited widening with range constraints and placing this widening at conditionals. The idea was first presented in [5] but has been refined, implemented and evaluated in this paper.

We have implemented both our and Simon & King's methods into the static analysis tool SWEET. The evaluation shows that in all the observed cases our method is at least as precise as Simon & King's, and in some cases is considerably better. This increase of precision is obtained without too much added analysis time.

Our method is particularly useful in the cases where the number of integer points inside polyhedra are interesting, since our method derives only polyhedra with a finite number of integer points. In particular it is interesting in the method for parametric worst-case execution time analysis developed in [15, 4], which is based on these numbers. We plan to evaluate our analysis for this application as well.

Acknowledgement. This work was supported by the EU FP7 project APARTS, Grant Number 251413.

References

1. Sweet (Feb 2012), http://www.mrtc.mdh.se/projects/wcet/sweet.html
2. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72(1–2), 3–21 (2008)
3. Brauer, J., King, A.: Transfer function synthesis without quantifier elimination. In: Barthe, G. (ed.) ESOP. Lecture Notes in Computer Science, vol. 6602, pp. 97–115. Springer (2011)
4. Bygde, S., Lisper, B.: Towards an automatic parametric WCET analysis. pp. 9–17. Austrian Computer Society (July 2008)
5. Bygde, S., Lisper, B., Holsti, N.: Fully bounded polyhedral analysis of integers with wrapping. In: International Workshop on Numerical and Symbolic Abstract Domains (NSAD'11) (September 2011), http://www.mrtc.mdh.se/index.php?choice=publications&id=2595
6. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL. pp. 238–252 (1977)
7. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL. pp. 84–96 (1978)
8. Granger, P.: Static analysis of arithmetical congruences. In: International Journal of Computer Mathematics, Volume 30. pp. 165–190 (1989)

9. Granger, P.: Static analysis of linear congruence equalities among variables of a program. In: Proceedings of the international joint conference on theory and practice of software development on Colloquium on trees in algebra and programming (CAAP '91): vol 1. pp. 169–192. Springer-Verlag New York, Inc., New York, NY, USA (1991), http://portal.acm.org/citation.cfm?id=111310.111320
10. Gustafsson, J., Betts, A., Ermedahl, A., Lisper, B.: The Mälardalen WCET benchmarks – past, present and future. pp. 137–147. OCG, Brussels, Belgium (Jul 2010)
11. Gustafsson, J., Ermedahl, A., Lisper, B.: Towards a flow analysis for embedded system C programs. In: The 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS05) (February 2005), http://www.mrtc.mdh.se/index.php?choice=publications&id=0972
12. Gustafsson, J., Ermedahl, A., Lisper, B., Sandberg, C., Källberg, L.: Alf a language for wcet flow analysis. In: Holsti, N. (ed.) Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET'09). OCG (June 2009), http://www.mrtc.mdh.se/index.php?choice=publications&id=1672
13. Gustafsson, J., Ermedahl, A., Sandberg, C., Lisper, B.: Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In: Proc. 27th IEEE Real-Time Systems Symposium (RTSS'06) (Dec 2006)
14. Halbwachs, N.: Delay analysis in synchronous programs. In: Courcoubetis, C. (ed.) CAV. Lecture Notes in Computer Science, vol. 697, pp. 333–346. Springer (1993)
15. Lisper, B.: Fully automatic, parametric worst-case execution time analysis. In: Gustafsson, J. (ed.) Proc. Third International Workshop on Worst-Case Execution Time (WCET) Analysis. pp. 77–80 (July 2003), http://www.mrtc.mdh.se/index.php?choice=publications& id=0629
16. Miné, A.: The octagon abstract domain. Higher Order Symbol. Comput. 19(1), 31–100 (2006)
17. Müller-Olm, M., Seidl, H.: Analysis of modular arithmetic. ACM Trans. Program. Lang. Syst. 29 (August 2007), http://doi.acm.org/10.1145/1275497.1275504
18. Sen, R., Srikant, Y.N.: Executable analysis using abstract interpretation with circular linear progressions. In: Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign. pp. 39–48. MEMOCODE '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/MEMCOD.2007.371251
19. Simon, A., King, A.: Taming the wrapping of integer arithmetic. In: Static Analysis, Lecture Notes in Computer Science, vol. 4634, pp. 121–136. Springer Berlin / Heidelberg (2007)
20. Verdoolaege, S., Seghir, R., Beyls, K., Loechner, V., Bruynooghe, M.: Counting integer points in parametric polytopes using Barvinok's rational functions. Algorithmica 48(1), 37–66 (Jun 2007), uRL: http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=41970, DOI: 10.1007/s00453-006-1231-0
21. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D.B., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P.P., Staschulat, J., Stenström, P.: The worst-case execution-time problem - overview of methods and survey of tools. ACM Trans. Embedded Comput. Syst. 7(3) (2008)