Solving Difference Constraints over Modular Arithmetic

Graeme Gange Harald Søndergaard
Peter J. Stuckey Peter Schachte

Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia
{ggange,harald,pjs,schachte}@csse.unimelb.edu.au

June 12, 2013

Outline

Program Analysis: A simple program

$$
\begin{array}{l}
x := \star \\
y := x \\
\textbf{for}(i := 0;\, i < 6;\, i := i + 1)\ \{ \\
\quad \textbf{if}(\star)\ y := y + 1 \\
\}
\end{array}
$$

Program Analysis: A simple program

$$
\begin{array}{l}
x := \star \\
y := x \\
\textbf{for}(i := 0;\ i < 6;\ i := i + 1)\ \{ \\
\quad \textbf{if}(\star)\ y := y + 1 \\
\}
\end{array}
$$

$$y - x \geq 0 \wedge x - y \geq -6$$

$$\equiv$$

$$0 \leq y - x \leq 6$$

Program Analysis: Two's complement

$uint\ x := \star$
$uint\ y := x$
$\textbf{for}(i := 0; i < 6; i := i + 1)\ \{$
$\quad \textbf{if}(\star)\ y := y + 1$
$\}$

$$0 \leq y - x \leq 6$$

Program Analysis: Two's complement

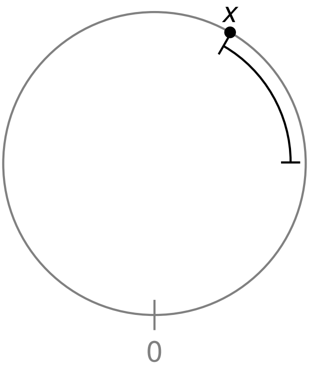
$$
\begin{array}{l}
\mathit{uint}\ x := \star \\
\mathit{uint}\ y := x \\
\textbf{for}(i := 0;\ i < 6;\ i := i + 1)\ \{ \\
\quad \textbf{if}(\star)\ y := y + 1 \\
\}
\end{array}
$$

$$0 \leq y - x \leq 6$$

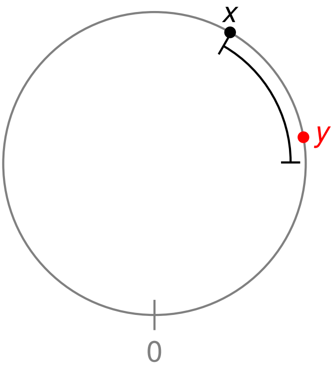
$$x = \mathbf{MAX_{uint}},\ y = 5$$

Well, that's awkward.

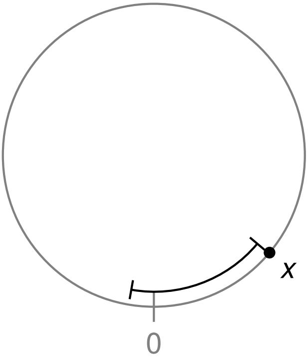
Program Analysis: Two's complement



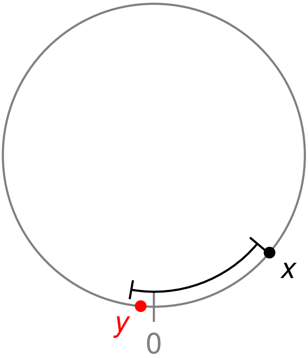
Program Analysis: Two's complement



Program Analysis: Two's complement



Program Analysis: Two's complement



Order and Proximity

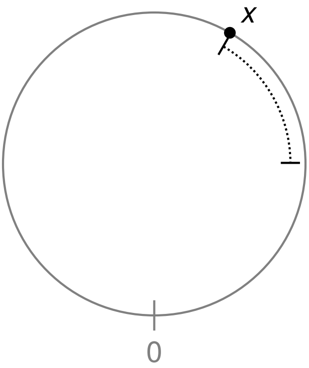
We need to distinguish between two kinds of relations:

- **Order** The numeric value of two numbers ($x \leq y$).
- **Proximity** Relative location on the number circle. ($y = x + 6$)

When reasoning over $\mathbb{Z}$, these two notions are equivalent.

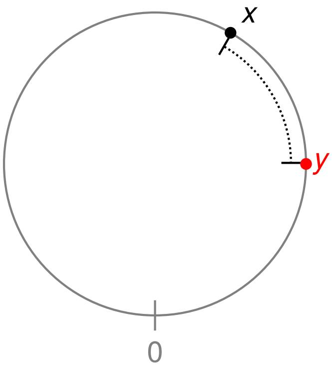
Reasoning about proximity

Notice that proximity constraints are always bounded on both sides.
Consider $y - x \in [6, \infty]$:



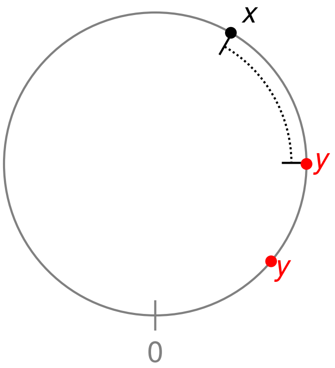
Reasoning about proximity

Notice that proximity constraints are always bounded on both sides.
Consider $y - x \in [6, \infty]$:



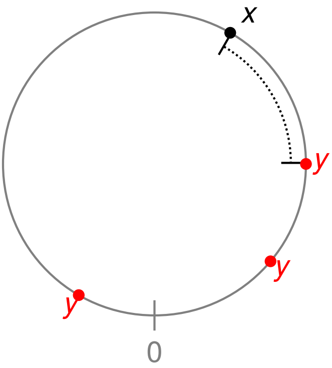
Reasoning about proximity

Notice that proximity constraints are always bounded on both sides.
Consider $y - x \in [6, \infty]$:



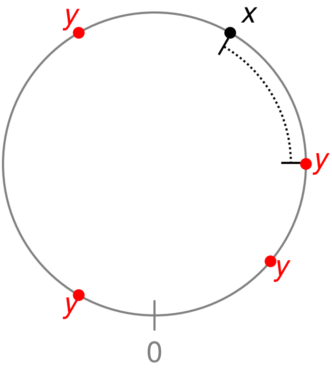
Reasoning about proximity

Notice that proximity constraints are always bounded on both sides.
Consider $y - x \in [6, \infty]$:

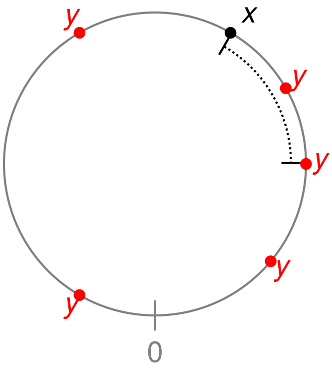


Reasoning about proximity

Notice that proximity constraints are always bounded on both sides.
Consider $y - x \in [6, \infty]$:



Notice that proximity constraints are always bounded on both sides.
Consider $y - x \in [6, \infty]$:



Reasoning about proximity

To reason about proximity constraints, we need to handle two kinds of inferences:

Resolution:

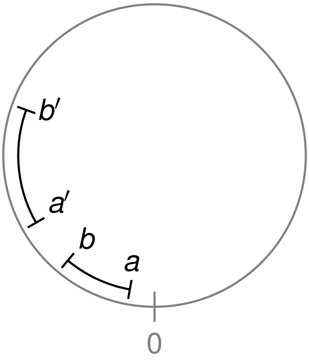
$$y - x \in [a, b] \wedge z - y \in [c, d] \models z - x \in ?$$

Intersection:

$$y - x \in [a, b] \wedge y - x \in [c, d] \models y - x \in ?$$

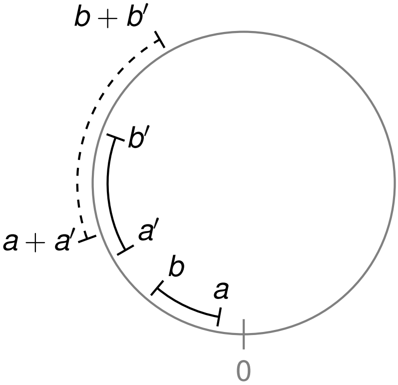
Resolution

We resolve pairs of constraints by adding the corresponding intervals:



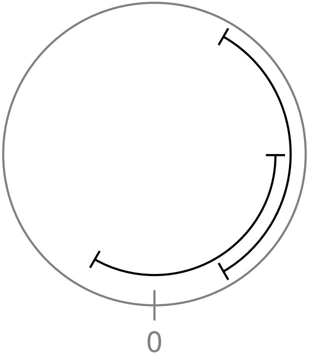
Resolution

We resolve pairs of constraints by adding the corresponding intervals:



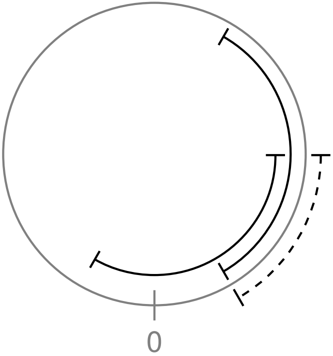
Intersection

We need to take the intersection of pairs of intervals:



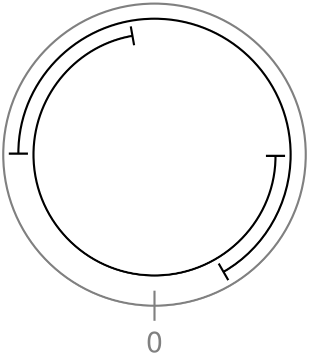
Intersection

We need to take the intersection of pairs of intervals:



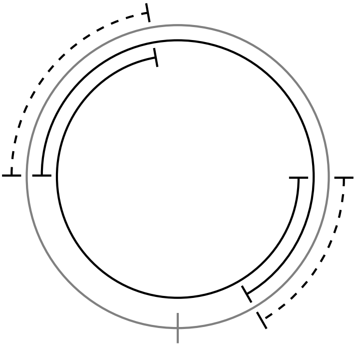
Intersection

...but we can't always represent it with a single interval.



Intersection

...but we can't always represent it with a single interval.



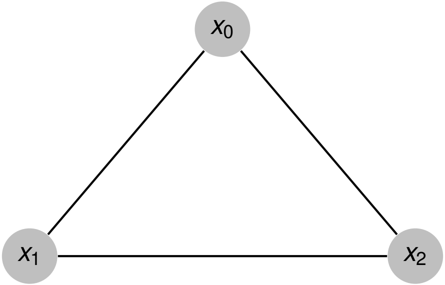
The bad news: Complexity

Satisfiability of a set of proximity constraints is NP-complete, even for small m.

The bad news: Complexity

Satisfiability of a set of proximity constraints is NP-complete, even for small m.

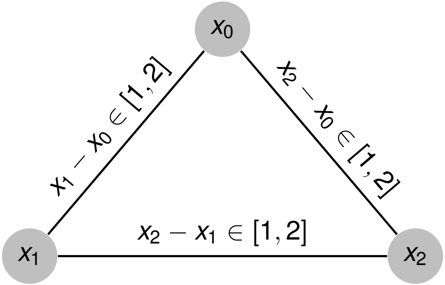
Reduction from 3-colouring:



The bad news: Complexity

Satisfiability of a set of proximity constraints is NP-complete, even for small m.

Reduction from 3-colouring:



A momentary diversion: Trade-offs

We care about 3 things:

- Correctness
- Precision
- Efficiency

A momentary diversion: Trade-offs

We care about 3 things:

- Correctness
- Precision
- Efficiency

Verification:

- We can trade time for additional precision.

Invariant Generation

- Precision is nice, but we can't spend *too* much time.

We *really* don't want to sacrifice soundness.

Satisfiability Modulo Theories (**SMT**)

SMT techniques are complete methods for families of NP-complete problems.

Two theories are of particular interest:

SMT$(\mathcal{BV})$ Bit-vectors

SMT$(\mathcal{DL})$ Difference logic

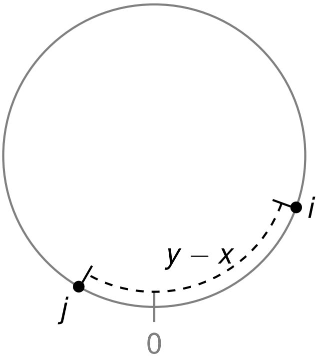
SMT($\mathcal{BV}$)

For $m = 2^b$, we can encode the machine arithmetic operations directly:

$$
\begin{array}{rcl}
x \leq y & \mapsto & x \leq_{\mathrm{u}} y \\
y - x \in [i, j] & \mapsto & (v_y \mathbin{-_{\mathtt{bv}}} v_x) \mathbin{-_{\mathtt{bv}}} i \leq_{\mathrm{u}} j \mathbin{-_{\mathtt{bv}}} i
\end{array}
$$

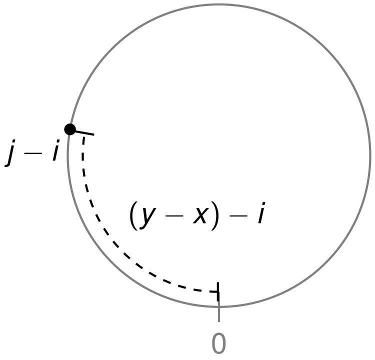
SMT($\mathcal{BV}$): $y - x \in [i, j]$

We can shift the number circle until the interval for $y - x$ starts at 0.



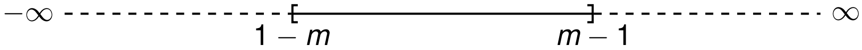
SMT($\mathcal{BV}$): $y - x \in [i, j]$

We can shift the number circle until the interval for $y - x$ starts at 0.



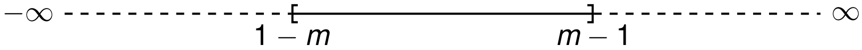
Mapping between wrapped and concrete values

Consider the range of $y - x$ (over $\mathbb{Z}$):

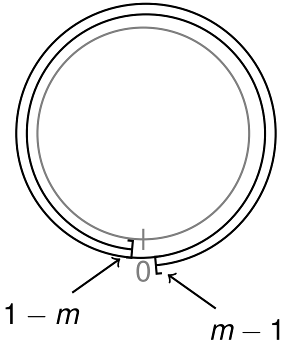


Mapping between wrapped and concrete values

Consider the range of $y - x$ (over $\mathbb{Z}$):

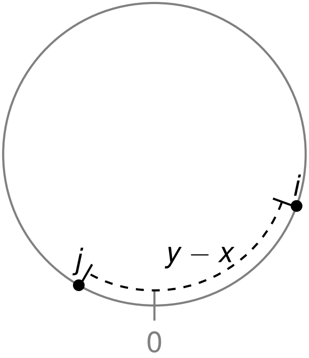


If we map it onto the number circle, we get:

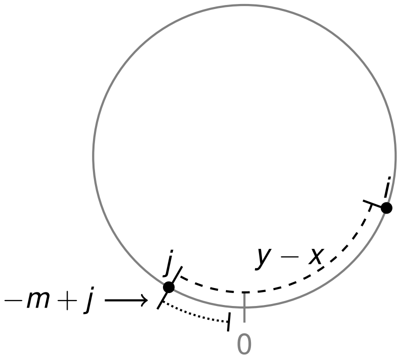


SMT($\mathcal{DL}$)

We can then encode a proximity constraint as a disjunction of classical difference constraints:

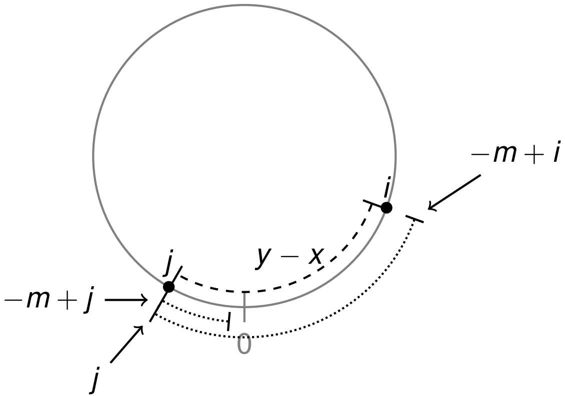


We can then encode a proximity constraint as a disjunction of classical difference constraints:



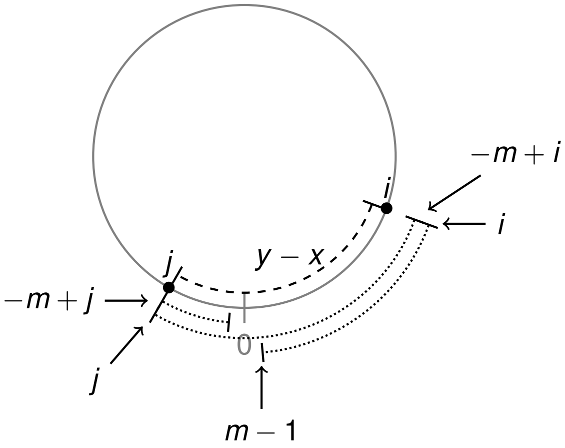
SMT($\mathcal{DL}$)

We can then encode a proximity constraint as a disjunction of classical difference constraints:



SMT($\mathcal{DL}$)

We can then encode a proximity constraint as a disjunction of classical difference constraints:



SMT($\mathcal{DL}$)

This yields the encoding:

$$
\begin{array}{rcl}
x \leq y & \mapsto & x \leq_{\mathrm{u}} y \\
y - x \in [i, j] & \mapsto & \begin{cases} \left(\begin{array}{ll} & -m + 1 \leq v_y - v_x \leq -m + j \\ \vee & -m + i \leq v_y - v_x \leq j \\ \vee & i \leq v_y - v_x \leq m - 1 \end{array} \right) & \textbf{if } j_m < i_m \\ \left(\begin{array}{ll} & -m + i \leq v_y - v_x \leq -m + j \\ \vee & i \leq v_y - v_x \leq j \end{array} \right) & \textbf{otherwise} \end{cases}
\end{array}
$$

Incomplete methods

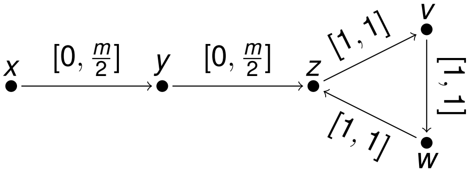
We *probably* don't want to be running an SMT solver in the inner loop of an abstract interpreter.

Can we adapt techniques from classical difference logic for a sound overapproximation?

The same basic idea: build a graph of constraints, and see if we can derive $\perp$.

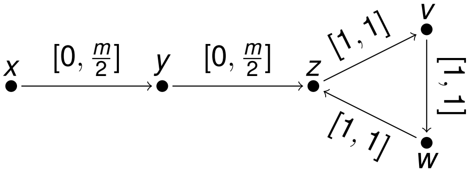
Incomplete methods

We can't use Bellman-Ford directly:



The path from x to z is already $\top$, so we never discover that $z \to v \to w \to z$ is inconsistent.

We can't use Bellman-Ford directly:

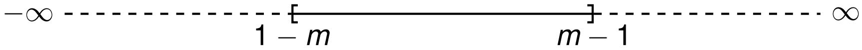


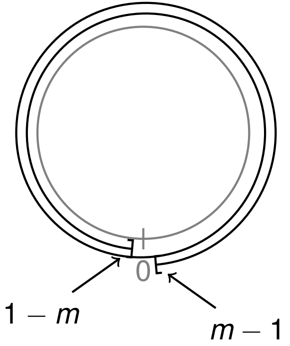
The path from x to z is already $\top$, so we never discover that $z \to v \to w \to z$ is inconsistent.

Floyd-Warshall is better, but a single iteration isn't guaranteed to reach a fixpoint.
Instead, we just apply a worklist algorithm until we can't tighten any constraints further.

Combining proximity and order

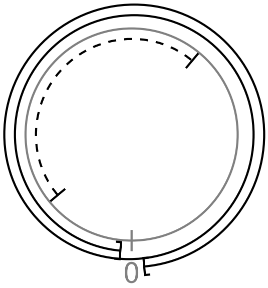
Recall the mapping of a concrete range onto the number circle:





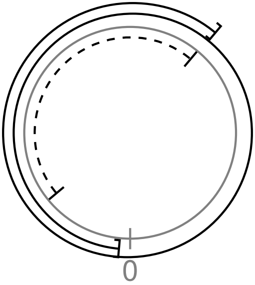
Combining proximity and order

Given a concrete and a wrapped interval, we can compute the reduced product:



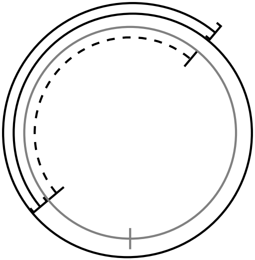
Combining proximity and order

Given a concrete and a wrapped interval, we can compute the reduced product:



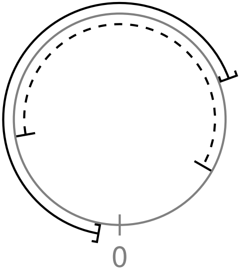
Combining proximity and order

Given a concrete and a wrapped interval, we can compute the reduced product:



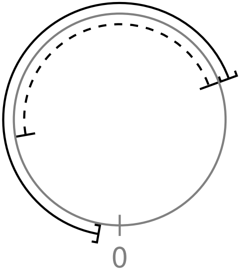
Combining proximity and order

Given a concrete and a wrapped interval, we can compute the reduced product:



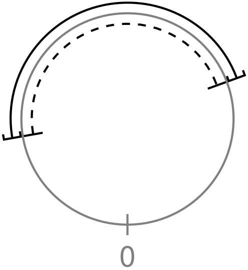
Combining proximity and order

Given a concrete and a wrapped interval, we can compute the reduced product:



Combining proximity and order

Given a concrete and a wrapped interval, we can compute the reduced product:



Experimental Results

Unfortunately, we don't (yet) have constraints from real programs.

Instead, we generated a range of random instances of increasing size:

- Fixed $|C| = 1.2|V|$
- 10% ordering constraints
- 100 instances of each size

Times are given in *ms*

Results: Random Instances

$\|V\|$	$\|C\|$	TIME$_{\mathcal{BV}}$	TIME$_{\mathcal{DL}}$	TIME$_{fix}$	#U	#FP
20	24	50.8	19.2	0.2	24	1
40	48	99.9	24.4	0.4	22	1
60	72	150.0	29.8	0.8	22	1
80	96	197.5	36.4	1.1	29	1
100	120	268.9	43.3	1.7	22	0
120	144	341.3	50.9	2.0	21	0
140	168	404.0	59.0	2.6	22	1
160	192	494.9	65.9	2.8	27	0
180	216	537.7	73.2	3.4	31	1
200	240	675.6	85.5	3.9	25	0