Welcome!

Decidable and undecidable problems

related to complexity analysis of loop programs.

Lars Kristiansen, Department of Mathematics, University of Oslo.
(Joint work with Amir Ben-Amram, Tel-Aviv.)

On the edge of decidability

We will discuss some problems on the edge of decidability.

On the edge of decidability

We will discuss some problems on the edge of decidability.

We have a problem $P_{a,b}$ where a and b are parameters.

On the edge of decidability

We will discuss some problems on the edge of decidability.

We have a problem $P_{a,b}$ where a and b are parameters.

We will play around with a and b and modify these parameters.

On the edge of decidability

We will discuss some problems on the edge of decidability.

We have a problem $P_{a,b}$ where a and b are parameters.

We will play around with a and b and modify these parameters.

This will alter status of the problem $P_{a,b}$ (e.g. from being undecidable to being decidable).

A table

constants:	none	0	0, 1
indefinite loops standard assignments	PTIME	PSPACE	?
definite loops weak assignments	undecidable	undecidable	undecidable
definite loops max-assignments	PTIME	PTIME	?

A table

constants:	none	0	0, 1
indefinite loops standard assignments	PTIME	PSPACE	?
definite loops weak assignments	undecidable	undecidable	undecidable
definite loops max-assignments	PTIME	PTIME	?

The aim of my talk is to explain this table.

Loop languages

$$
\begin{array}{rcl}
\mathtt{X}, \mathtt{Y}, \mathtt{Z} \in \text{Variable} & ::= & \mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots \mid \mathtt{X}_n \\
\mathtt{C} \in \text{Command} & ::= & \mathtt{X{:=}Y} \mid \mathtt{X{:=}Y{+}Z} \mid \mathtt{X{:=}0} \mid \mathtt{X{:=}Y{+}1} \\
 & & \mid \mathtt{C}_1\mathtt{;}\mathtt{C}_2 \mid \mathtt{!loop\ X\ \{C\}}
\end{array}
$$

This is the syntax of a typical loop-language.

Loop languages

$$
\begin{array}{rcl}
\mathtt{X}, \mathtt{Y}, \mathtt{Z} \in \text{Variable} & ::= & \mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots \mid \mathtt{X}_n \\
\mathtt{C} \in \text{Command} & ::= & \mathtt{X{:=}Y} \mid \mathtt{X{:=}Y{+}Z} \mid \mathtt{X{:=}0} \mid \mathtt{X{:=}Y{+}1} \\
 & & \mid \mathtt{C}_1\mathtt{;}\mathtt{C}_2 \mid \mathtt{!loop\ X}\ \{\mathtt{C}\}
\end{array}
$$

This is the syntax of a typical loop-language.

The semantics is obvious to anyone familiar with imperative programming languages.

Loop languages

$$
\begin{array}{rcl}
\mathtt{X}, \mathtt{Y}, \mathtt{Z} \in \text{Variable} & ::= & \mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots \mid \mathtt{X}_n \\
\mathtt{C} \in \text{Command} & ::= & \mathtt{X{:=}Y} \mid \mathtt{X{:=}Y{+}Z} \mid \mathtt{X{:=}0} \mid \mathtt{X{:=}Y{+}1} \\
 & & \mid \mathtt{C}_1\mathtt{;}\mathtt{C}_2 \mid \mathtt{!loop\ X}\ \{\mathtt{C}\}
\end{array}
$$

This is the syntax of a typical loop-language.

The semantics is obvious to anyone familiar with imperative programming languages.

We will call $\mathtt{!loop\ X}\ \{\mathtt{C}\}$ a definite loop.

Loop languages

$$
\begin{array}{rcl}
\mathtt{X}, \mathtt{Y}, \mathtt{Z} \in \text{Variable} & ::= & \mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots \mid \mathtt{X}_n \\
\mathtt{C} \in \text{Command} & ::= & \mathtt{X:=Y} \mid \mathtt{X:=Y+Z} \mid \mathtt{X:=0} \mid \mathtt{X:=Y+1} \\
 & & \mid \mathtt{C}_1\mathtt{;C}_2 \mid \mathtt{!loop\ X\ \{C\}}
\end{array}
$$

This is the syntax of a typical loop-language.

The semantics is obvious to anyone familiar with imperative programming languages.

We will call $\mathtt{!loop\ X\ \{C\}}$ a definite loop.

$\mathtt{!loop\ X\ \{C\}} \equiv \underbrace{\mathtt{C;\ C;\ .....;\ C}}_{n}$ where $\mathtt{X} \mapsto n$.

The feasibility problem

We will consider the *the feasibility problem* for several variants of such a loop language.

The feasibility problem

We will consider the *the feasibility problem* for several variants of such a loop language.

What is the feasibility problem?

The feasibility problem

We will consider the *the feasibility problem* for several variants of such a loop language.

What is the feasibility problem?

Definition. Let $\mathtt{p}$ be a program over the variables $\mathtt{X}_1, \ldots, \mathtt{X}_n$, and let $x_1, \ldots, x_n$ be the values of respectively $\mathtt{X}_1, \ldots, \mathtt{X}_n$ when an execution of $\mathtt{p}$ starts. The program $\mathtt{p}$ is *feasible* if there exists a polynomial p such that $p(x_1, \ldots, x_n)$ bounds any value computed during the execution. We define the *feasibility problem for the language L* by

- input: an L-program $\mathtt{p}$; question: is $\mathtt{p}$ feasible?

Two simple examples

This is a feasible program:

```
!loop X { !loop Y { !loop Z { !loop U { W:= W+1 } } } }
```

Two simple examples

This is a feasible program:

```
!loop X { !loop Y { !loop Z { !loop U { W:= W+1 } } } }
```

This is a *not* feasible program:

```
!loop X { Y:= Y+Y }
```

Decidable and undecidable problems

In general, it is of course very hard to decide if an arbitrary program written in the loop language given above is feasible or not.

Decidable and undecidable problems

In general, it is of course very hard to decide if an arbitrary program written in the loop language given above is feasible or not.

It should be no surprise that

the feasibility problem is undecidable

when we are working with a standard loop language.

Decidable and undecidable problems

But it is a bit surprising that

the feasibility problem is decidable

when we are dealing with a loop language with *indefinite* loops.

What is an *indefinite* loop?

What is an *indefinite* loop?

Recall the *definite loop*:

$\texttt{!loop X } \{\texttt{C}\} \equiv \underbrace{\texttt{C; C; .....; C}}_{n}$ where $\texttt{X} \mapsto n$.

The number held by `X` gives the exact number of times the loop's body will be executed.

What is an *indefinite* loop?

Recall the *definite loop*:

$\texttt{!loop X } \{\texttt{C}\} \equiv \underbrace{\texttt{C; C; .....; C}}_{n}$ where $\texttt{X} \mapsto n$.

The number held by `X` gives the exact number of times the loop's body will be executed.

This is the *indefinite loop*:

$\texttt{?loop X } \{\texttt{C}\} \equiv \underbrace{\texttt{C; C; .....; C}}_{k}$ where $\texttt{X} \mapsto n \geq k$.

The number held by `X` gives an *upper bound* on the number of times the loop's body will be executed. (We are dealing with non-determinism.)

What is an *indefinite* loop?

Recall the *definite loop*:

$\texttt{!loop X}\ \{\texttt{C}\} \equiv \underbrace{\texttt{C; C; .....; C}}_{n}$ where $\texttt{X} \mapsto n$.

The number held by `X` gives the exact number of times the loop's body will be executed.

This is the *indefinite loop*:

$\texttt{?loop X}\ \{\texttt{C}\} \equiv \underbrace{\texttt{C; C; .....; C}}_{k}$ where $\texttt{X} \mapsto n \geq k$.

The number held by `X` gives an *upper bound* on the number of times the loop's body will be executed. (We are dealing with non-determinism.)

The loop's body may be executed zero times? once? twice? three times? ... but not more than n times!

Decidable and undecidable problems

Now, assume we are dealing with loop languages where the loops are indefinite.

Then, the feasibility problem is *decidable*

Decidable and undecidable problems

Now, assume we are dealing with loop languages where the loops are indefinite.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`

Decidable and undecidable problems

Now, assume we are dealing with loop languages where the loops are indefinite.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PSPACE* (complete)
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

Decidable and undecidable problems

Now, assume we are dealing with loop languages where the loops are indefinite.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PSPACE* (complete)
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

These two problems are still decidable (in resp. *PTIME* and *PSPACE*) if we also allow `X:=Y*Z`.

Decidable and undecidable problems

Now, assume we are dealing with loop languages where the loops are indefinite.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PSPACE* (complete)
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

These two problems are still decidable (in resp. *PTIME* and *PSPACE*) if we also allow `X:=Y*Z`.

But it is now know if the problem is decidable if we also allow

Decidable and undecidable problems

Now, assume we are dealing with loop languages where the loops are indefinite.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PSPACE* (complete)
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

These two problems are still decidable (in resp. *PTIME* and *PSPACE*) if we also allow `X:=Y*Z`.

But it is now know if the problem is decidable if we also allow `X:=1`.

The table again

constants:	none	0	0, 1
indefinite loops standard assignments	**PTIME**	**PSPACE**	**?**
definite loops weak assignments	undecidable	undecidable	undecidable
definite loops max assignments	PTIME	PTIME	?

Thus, I have explained the first row of my table.

. . . and again

constants:	none	0	0, 1
indefinite loops standard assignments	PTIME	PSPACE	?
definite loops **weak assignment**	**undecidable**	**undecidable**	**undecidable**
definite loops max assignments	PTIME	PTIME	?

I will now turn to second row.

Determinism and non-determinism

We have seen that the the feasibility problem is decidable when we have

- indefinite loops `?loop X {C}`
  These are *non-deterministic* loops.
- standard assignments, e.g. `X:=Y+Z` and `X:=0`.
  These are *deterministic* assignments.

Determinism and non-determinism

We have seen that the the feasibility problem is decidable when we have

- indefinite loops `?loop X {C}`
  These are *non-deterministic* loops.
- standard assignments, e.g. `X:=Y+Z` and `X:=0`.
  These are *deterministic* assignments.

The decidability seems to be due to the non-deterministic loops.

Determinism and non-determinism

We have seen that the the feasibility problem is decidable when we have

- indefinite loops `?loop X {C}`
  These are *non-deterministic* loops.
- standard assignments, e.g. `X:=Y+Z` and `X:=0`.
  These are *deterministic* assignments.

The decidability seems to be due to the non-deterministic loops.

What happens if we swap and have some kind of non-deterministic assignments, but standard deterministic loops?

Determinism and non-determinism

We have seen that the the feasibility problem is decidable when we have

- indefinite loops `?loop X { C }`
  These are *non-deterministic* loops.
- standard assignments, e.g. `X:=Y+Z` and `X:=0`.
  These are *deterministic* assignments.

The decidability seems to be due to the non-deterministic loops.

What happens if we swap and have some kind of non-deterministic assignments, but standard deterministic loops?

We have investigated *weak assignments*

- $\texttt{X} :\leq \langle exp \rangle$

where the value of `X` is set to some natural number $\leq \langle exp \rangle$.

Decidable and undecidable problems

We were surprised to discover that the feasibility problem is *undecidable* when the language have

- standard definite (i.e. deterministic) loops
- weak (i.e. non-deterministic) assignments.

This holds even if we only allow assignments in the form `X` $:\leq$ `Y+Z`. (This language seems very weak. Still, the feasibility problem is undecidable.)

The table

constants:	none	0	0, 1
indefinite loops standard assignments	PTIME	PSPACE	?
definite loops **weak assignment**	**undecidable**	**undecidable**	**undecidable**
definite loops max assignments	PTIME	PTIME	?

This explains the second row.

The table

constants:	none	0	0, 1
indefinite loops standard assignments	PTIME	PSPACE	?
definite loops weak assignment	undecidable	undecidable	undecidable
definite loops **max-assignments**	**PTIME**	**PTIME**	**?**

Let us look at the third row.

The table

constants:	none	0	0, 1
indefinite loops standard assignments	PTIME	PSPACE	?
definite loops weak assignment	undecidable	undecidable	undecidable
definite loops **max-assignments**	**PTIME**	**PTIME**	**?**

Let us look at the third row.

Here we also have the standard deterministic loops.

The table

constants:	none	0	0, 1
indefinite loops standard assignments	PTIME	PSPACE	?
definite loops weak assignment	undecidable	undecidable	undecidable
definite loops **max-assignments**	**PTIME**	**PTIME**	**?**

Let us look at the third row.

Here we also have the standard deterministic loops.

And we also have deterministic statements. But we do not have standard assignments, we have so-called max-assignments $\mathtt{X} :\overset{\max}{=} \langle exp \rangle$.

What is an max-assignment?

The assignment

$$\texttt{X} :\overset{\max}{=} \langle exp \rangle$$

sets `X` to the value of $\langle exp \rangle$ if this increases the value of `X`; otherwise `X` keeps its current value.

What is an max-assignment?

The assignment

$$\texttt{X} :\overset{\max}{=} \langle exp \rangle$$

sets `X` to the value of $\langle exp \rangle$ if this increases the value of `X`; otherwise `X` keeps its current value.

These assignments restore the decidability of the feasibility problem.

Decidable and undecidable problems

Assume we are dealing with loop languages with standard definite loops and max-assignments.

Then, the feasibility problem is *decidable*

Decidable and undecidable problems

Assume we are dealing with loop languages with standard definite loops and max-assignments.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`

Decidable and undecidable problems

Assume we are dealing with loop languages with standard definite loops and max-assignments.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PTIME*
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

Decidable and undecidable problems

Assume we are dealing with loop languages with standard definite loops and max-assignments.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PTIME*
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

These two problems are still decidable (in *PTIME*) if we also allow $\texttt{X}:\overset{\max}{=}\texttt{Y}*\texttt{Z}$.

Decidable and undecidable problems

Assume we are dealing with loop languages with standard definite loops and max-assignments.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PTIME*
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

These two problems are still decidable (in *PTIME*) if we also allow $\texttt{X}:\overset{\max}{=}\texttt{Y*Z}$.

We have not yet proved that the problem is decidable if we also allow `X:=1`.

Decidable and undecidable problems

Assume we are dealing with loop languages with standard definite loops and max-assignments.

Then, the feasibility problem is *decidable*

- in *PTIME*
  if we only allow assignments of the forms `X:=Y` and `X:=Y+Z`
- in *PTIME*
  if we only allow the `X:=Y` and `X:=Y+Z` and `X:=0`.

These two problems are still decidable (in *PTIME*) if we also allow $\texttt{X:}\overset{\max}{=}\texttt{Y*Z}$.

We have not yet proved that the problem is decidable if we also allow `X:=1`. (But I will be surprised if it isn't.)

The table

Thus, I have explained my table.

constants:	none	0	0, 1
indefinite loops standard ass.	PTIME	PSPACE	?
definite loops weak ass.	undecidable	undecidable	undecidable
definite loops max-ass.	PTIME	PTIME	?

The table

Thus, I have explained my table.

constants:	none	0	0, 1
indefinite loops standard ass.	PTIME	PSPACE	?
definite loops weak ass.	undecidable	undecidable	undecidable
definite loops max-ass.	PTIME	PTIME	?

In our papers you will also find the description of a few other related problems being on the edge of decidability.

Do I have some time left?

If so, let me explain way we became interested in these problems?

Do I have some time left?

If so, let me explain way we became interested in these problems?

We was (are) interested in static complexity analysis of programs.

Do I have some time left?

If so, let me explain way we became interested in these problems?

We was (are) interested in static complexity analysis of programs.

We want to analyze natural programs written in widely used programming like C and Java.

Do I have some time left?

If so, let me explain way we became interested in these problems?

We was (are) interested in static complexity analysis of programs.

We want to analyze natural programs written in widely used programming like C and Java.

The goal is to automatically verify complexity properties of such programs (e.g. does a program run in polynomial time).

References

The results I have presented are already published:

- **Amir M. Ben-Amram, Neil D. Jones, and Lars Kristiansen:** Linear, polynomial or exponential? complexity inference in polynomial time. In *Logic and Theory of Algorithms, CiE 2008*, volume 5028 of *LNCS*, pages 67–76. Springer, 2008.
- **Amir M. Ben-Amram:** On decidable growth-rate properties of imperative programs. In *International Workshop on Developments in Implicit Computational complExity (DICE 2010)*, volume 23 of *EPTCS*, pages 1–14. ArXiv.org, 2010.
- **Amir M. Ben-Amram and Lars Kristiansen:** *On the Edge of Decidability in Complexity Analysis of Loop Programs.* International Journal of Foundations of Computer Science (accepted).

Thanks for you attention!

This talk was based on joint work with

- Amir M. Ben-Amram (Tel-Aviv).