Integration of Offline Partial Deduction and Functional Conversion for miniKanren

Aleksandr Shefer
Constructor University
Bremen, Germany
JetBrains Research
Germany
alex.shefer.31@gmail.com

Ekaterina Verbitskaia
Constructor University
Bremen, Germany
JetBrains Research
Amsterdam, Netherlands
kajigor@gmail.com

In pure logic programming, a program is viewed as a relation that can be queried in different directions to find solutions for multiple problems. For example, an interpreter can both evaluate and synthesize programs. Unfortunately, the performance of the queries is a great concern, and sometimes it is impossible to create a relation that executes in all directions in a reasonable time. Program transformation techniques such as partial deduction and functional conversion have previously shown their potential when the execution direction of a relation is known. In this extended abstract, we present the integration of offline partial deduction with functional conversion, capable of greater performance improvement compared to the transformations run individually.

1 Introduction

Logic programming facilitates a relational style in which programs do not distinguish between input and output arguments [**?**]. Any program can solve many problems by being evaluated in different directions, notably forward and backward. Employing a complete search strategy, such as the interleaving search of KANREN[1], and abstaining from using extra-logical constructs such as cuts allows producing every solution to a query in any direction [**?**]. This approach, called relational programming, is capable of producing solvers by running a relational interpreter backwards [**?**]. For instance, a relational sorting algorithm can generate permutations while a program interpreter can be inverted to become a synthesizer.

The main issue with the approach is its inefficiency. Relational interpreters are either generated by relational conversion from a function or crafted by a person who has a specific, usually forward, direction in mind. In both cases, the interpreter often shows reasonable performance in the forward direction, but not necessarily when executed in any other direction. Besides, logic languages come with additional overhead, such as expensive unification with occurs-check. Finally, being an embedded domain-specific language, KANREN comes with an interpretation overhead which also affects performance.

In has been shown that partial deduction can alleviate some of the interpretation overhead [**?**, **?**]. Additionally, functional conversion is capable of improving the execution time of a program by translating it into a functional language in the context of a given direction [**?**]. While doing so, costly logical operations are replaced with much cheaper functional counterparts. We believe that combining the two approaches together can lead to even bigger speed-ups.

This extended abstract focuses on integrating an offline partial deduction algorithm with a functional conversion for a MINIKANREN[2] programming language. We illustrate the approach with an example and provide the preliminary evaluation that demonstrates noticeable execution time improvement.

[1] Kanren is a family of pure relational programming languages, website: `http://minikanren.org`
[2] MINIKANREN is a minimal language in the KANREN family

Submitted to:
HCVS 2024

```
let rec sort^o x y =
    (x ≡ [] ∧ y ≡ []) ∨
    (fresh s, xs, xs1 in
        (smallest^o x s xs ∧
         sort^o xs xs1 ∧
         y ≡ (s :: xs1)))
```

(a) Implementation in MINIKANREN

```
let sort^o_OI x =
    (x ≡ (0 :: 1 :: 2 :: []) ∨
     x ≡ (0 :: 2 :: 1 :: []) ∨
     x ≡ (1 :: 2 :: 0 :: []) ∨ ...)
```

(b) Result of the partial deduction of the call `sorto x (0 :: 1 :: 2 :: [])`

Figure 1: Sorting relation

2 Combining Specialization Methods for Relational Programming

Knowing values of some arguments of a relation or its direction enables specialization—an automatic program transformation aimed at improving the program's execution time [**?**]. Previously, two approaches were explored in the context of MINIKANREN: online partial deduction [**?**] and functional conversion [**?**]. The former symbolically executes a relation when the values or shape of some of its arguments are known at specialization-time. The latter converts a relational program into a function which produces the same results, but is devoid of costly operations inherent to a relational language, such as unification.

Consider the relation $\texttt{sort}^o$ which relates the list `x` with its sorted counterpart `y`. The implementation follows directly from the definition of a sorted list: it is either empty or its head is the smallest element of the list, and its tail is sorted. This is encoded as the disjunction ($\vee$) of two conjunctions ($\wedge$) which unify ($\equiv$) the sorted list with the empty and non-empty list constructors correspondingly. The relation $\texttt{smallest}^o$ `x s xs` relates the list `x`, its smallest element `s` and the list of remaining elements `xs`.

If it is known that the relation $\texttt{sort}^o$ is always going to be executed on lists of a certain length, say 10, $\texttt{sort}^o$ will only be called a finite number of times (11, to be exact). Moreover, only one of the two disjuncts will succeed on every call. A partial deduction algorithm will unfold a body of the relation the appropriate number of times and partially evaluate it, thereby removing some of the unnecessary computations. More sophisticated techniques, such as Conjunctive Partial Deduction [**?**], include the information about variable sharing or run additional analyses and use it in the transformations.

Functional conversion translates the relation into a function in the context of a specific direction. While doing so, it replaces expensive unifications with much cheaper equality checks, variable bindings or term construction operations—based on which role the unification serves in the given direction. For example, if we know that $\texttt{sort}^o$ is run with the second argument known, then unifications of `y` may be replaced with a pattern match. One peculiar thing about $\texttt{sort}^o$ is that it is impossible to write a single program which behaves well in both directions. Depending on the order of calls to $\texttt{smallest}^o$ and $\texttt{sort}^o$ it will quickly time out when sorting or when generating permutations. Using mode analysis, functional conversion puts the calls in the order best suited for the particular direction.

Both transformations are capable of achieving significant results; combining them together is the next logical step. But we first had to address a couple of shortcomings of the previous works. Online partial deduction tends to produce extremely large programs due to making all the decisions at specialization-time without running any analysis beforehand. Apart from that, partial deduction will benefit from knowing the order of calls, which can be done by employing mode analysis implemented as a part of functional conversion. As a result, we decided to switch to offline partial deduction that runs static analyses such as binding time analysis, termination check and mode inference before partial evaluation.

		Translation with CPD		Functional Conversion
		Online	Offline	
sorto	`[0;1;2;3](5 answers)`	174.00 µs	0.15 µs	26.90 µs
sorto	`[0;...;4](5 answers)`	20 600.00 µs	0.19 µs	183.00 µs
hanoi	`[3 disks](5 answers)`	125 000.00 µs	1.67 µs	153 000.00 µs
hanoi	`[4 disks](5 answers)`	timeout	3.12 µs	timeout

Figure 2: Evaluation results of relational sorting

The resulting specialization algorithm is done in five steps. First, the user annotates the relation with its intended direction, namely which arguments are input and which are output. Then the relation is mode analysed, which reorders the conjuncts for the direction and annotates the other calls in the program. After that, a simple binding time analysis propagates the information about input arguments. It employs termination check to annotate which of the relation calls need to be unfolded at specialization-time. Next, partial deduction uses the results of the binding time analysis and mode analysis to partially evaluate the program. Finally, the resulting program in MINIKANREN is converted into a function in HASKELL.

We provide the result of the partial deduction of $\texttt{sort}^o\ \texttt{x}\ (0 :: 1 :: 2 :: [])$ in Figure 1b. Our approach successfully unfolds the program completely, resulting in the enumeration of all possible answers to the query. Note that, depending on the relation, direction and the known values of the arguments, it is not always possible.

3 Evaluation

At this point, we have only done a limited evaluation of our approach. Here we present the comparison of the described transformation with functional conversion and the online partial deduction algorithm. For that, we run the corresponding methods on two examples and compared the execution time.

The first benchmark is $\texttt{sort}^o$ specialized for the backwards direction in which it generates all permutations of a given list. In this case, our approach results in a program which enumerates all possible answers, while both online partial deduction and functional conversion fail to achieve this degree of specialization. The number of permutations grows rapidly as the list's length increases, so we only run the function on the list of length up to five. Querying it for the first five answers demonstrates the performance increase of 3 orders of magnitude compared with functional conversion and up to 5 orders of magnitude compared with online partial deduction: see first two rows in Table 2.

The second benchmark is the classic Tower of Hanoi puzzle. It is a representative program for verifier-to-solver approach. The program is given a sequence of disks' moves and checks if the finish state can be achieved from the start state by it. Initially, the program was written in a functional language and translated into MINIKANREN, then it was specialized in the direction which generates the sequence of steps, solving the puzzle. The result of querying for 5 possible answers is at least 5 orders of magnitude faster than both functional conversion and online partial deduction: see the last two rows in Table 2.

4 Conclusion and Future work

In this abstract, we described the integration of offline partial deduction with functional conversion for MINIKANREN. Our preliminary evaluation showed solid performance improvement. We believe that it is a promising direction which is worth further investigation.

References

[1] Danny De Schreye, Robert Glück, Jesper Jørgensen, Michael Leuschel, Bern Martens & Morten Heine Sørensen (1999): *Conjunctive partial deduction: Foundations, control, algorithms, and experiments*. *The Journal of Logic Programming* 41(2-3), doi:10.1016/S0743-1066(99)00030-8.

[2] Daniil Berezun Ekaterina Verbitskaia, Igor Engel (2024): *A Case Study in Functional Conversion and Mode Inference in miniKanren*. *PEPM 2024: Proceedings of the 2024 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, doi:10.1145/3635800.3636966.

[3] Daniel P. Friedman, William E. Byrd & Oleg Kiselyov (2005): *The Reasoned Schemer*. The MIT Press, doi:10.7551/mitpress/5801.001.0001.

[4] John P. Gallagher (1993): *Tutorial on Specialisation of Logic Programs*. In: *Proceedings of the 1993 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, PEPM '93, New York, NY, USA, pp. 88–98, doi:10.1145/154630.154640.

[5] Oleg Kiselyov, Chung-chieh Shan, Daniel P Friedman & Amr Sabry (2005): *Backtracking, interleaving, and terminating monad transformers: (functional pearl)*. *ACM SIGPLAN Notices* 40(9), pp. 192–203.

[6] Petr Lozov, Ekaterina Verbitskaia & Dmitry Boulytchev (2019): *Relational interpreters for search problems*. In: *Relational Programming Workshop*, p. 43.

[7] Ekaterina Verbitskaia, Daniil Berezun & Dmitry Boulytchev (2021): *An Empirical Study of Partial Deduction for miniKanren*. In Alexei Lisitsa & Andrei P. Nemytykh, editors: Proceedings of the 9th International Workshop on *Verification and Program Transformation*, Luxembourg, Luxembourg, 27th and 28th of March 2021, *Electronic Proceedings in Theoretical Computer Science* 341, Open Publishing Association, pp. 73–94, doi:10.4204/EPTCS.341.5.