

MRes Dissertation

Pattern Matching Strategies for Peephole Optimisation

Elif Aktolga

Supervisor: Dr. Des Watson

- Introduction: Peephole Optimisation (PO)
- Motivation
- Existing Approaches (PO, Pattern Matching)
- PO as an Information Processing System
- Pattern Matching Strategies (declarative, generic)
- Cascading Rules Strategy
- Results
- Summary
- Outlook

- Peephole Optimisation (PO): post-compilation optimisation technique used in compilers (last stages)
- **aim:** finding inefficient sequences of code (mostly produced during code generation)

- Peephole Optimisation (PO): post-compilation optimisation technique used in compilers (last stages)
- **aim:** finding inefficient sequences of code (mostly produced during code generation)
- mostly applied to **assembly code** with **optimisation rules**
- optimisation rules: matching part + replacement part

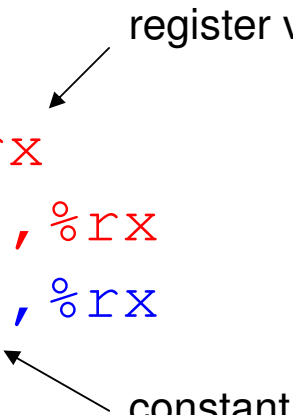
- Peephole Optimisation (PO): post-compilation optimisation technique used in compilers (last stages)
- **aim:** finding inefficient sequences of code (mostly produced during code generation)
- mostly applied to **assembly code** with **optimisation rules**
- optimisation rules: matching part + replacement part
- peephole: small window → analysis of small sequences of code with the rules → replacements with rules whenever inefficient code is found
- repeated optimisation until no more improvements possible

Example:

Rule: `clr %rx`
`mov n1, %rx`
 \Rightarrow `mov n1, %rx`

register variable

constant



Example:

Rule: `clr %rx` → IF these **match**
`mov n1, %rx` → THEN **replace** with this
=> `mov n1, %rx`

register variable

constant

Example:

Rule: `clr %rx` → IF these **match**
`mov n1, %rx` → THEN **replace** with this
=> `mov n1, %rx`

register variable

constant

Code: `clr %r0`
`mov 1, %r0`
`cmp 2, %r0`

Example:

Rule: `clr %rx` → IF these **match**
`mov n1, %rx` → THEN **replace** with this
=> `mov n1, %rx`

register variable

constant

Code: `clr %r0` Optimised: `mov 1, %r0`
`mov 1, %r0` `cmp 2, %r0`
`cmp 2, %r0`

- PO: traditionally done by **string pattern matching** (regular expressions)

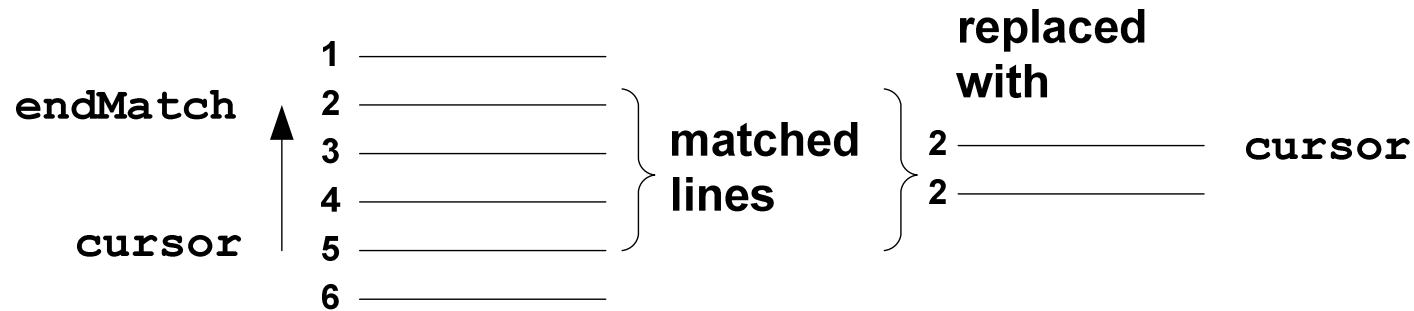
- PO: traditionally done by **string pattern matching** (regular expressions)
- **problem:** successful; but the assembly code is treated as a collection of strings only → limited optimisation

- PO: traditionally done by **string pattern matching** (regular expressions)
- **problem:** successful; but the assembly code is treated as a collection of strings only → limited optimisation
- **research question: Can PO be achieved even more successfully by utilising the meaning of the code? How?**

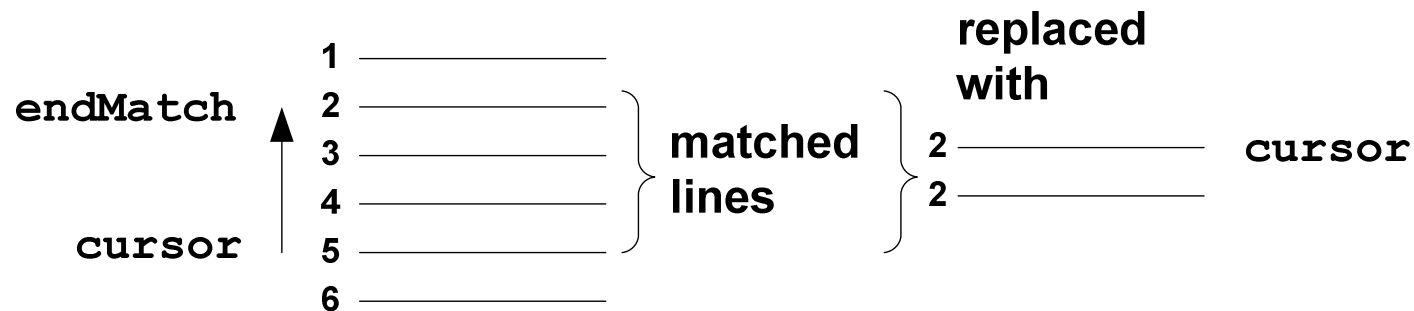
- PO: traditionally done by **string pattern matching** (regular expressions)
 - **problem:** successful; but the assembly code is treated as a collection of strings only → limited optimisation
 - **research question: Can PO be achieved even more successfully by utilising the meaning of the code? How?**
- **goal:** exploration of pattern matching techniques:
Alternatives to matching by regular expressions

- PO: traditionally done by **string pattern matching** (regular expressions)
- **problem:** successful; but the assembly code is treated as a collection of strings only → limited optimisation
- **research question: Can PO be achieved even more successfully by utilising the meaning of the code? How?**
- **goal:** exploration of pattern matching techniques:
Alternatives to matching by regular expressions
- Remark: PO is an old technique from the 1980s → Can it be combined with newer concepts & approaches like OOP?

- Lamb (1981) describes the implementation of a PO system with regular expressions and a **backwards strategy**:

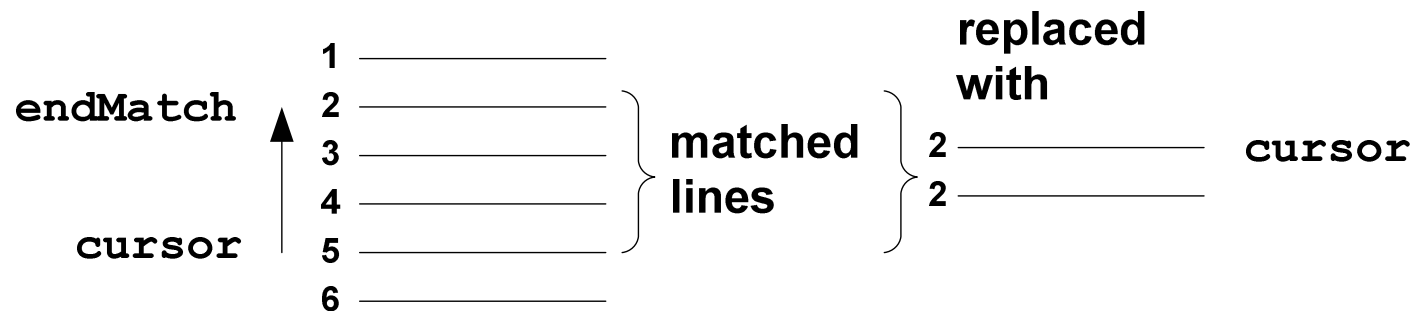


- Lamb (1981) describes the implementation of a PO system with regular expressions and a **backwards strategy**:



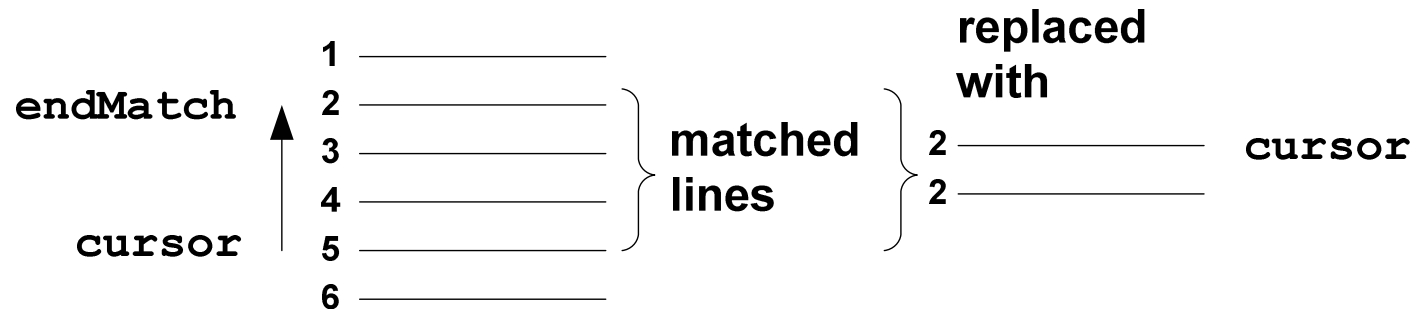
- determines how **optimisation rules are applied**: `cursor` moves forward, but rules are matched with preceding lines

- Lamb (1981) describes the implementation of a PO system with regular expressions and a **backwards strategy**:



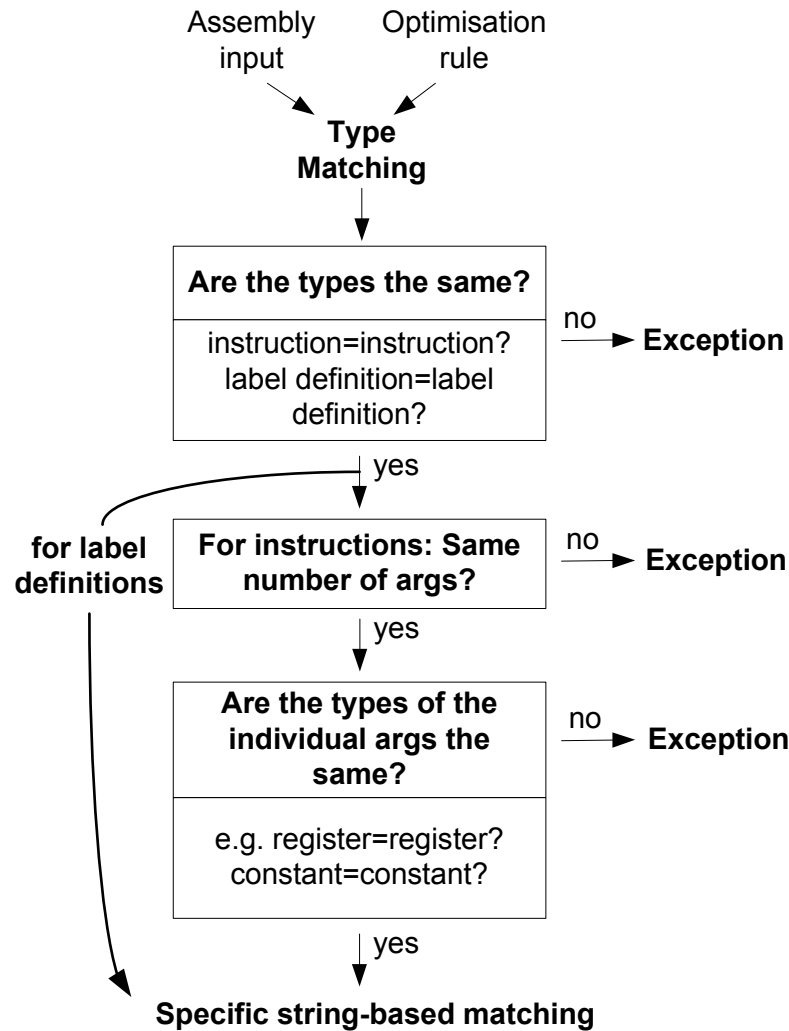
- determines how **optimisation rules are applied**: `cursor` moves forward, but rules are matched with preceding lines
→ optimised lines can **immediately** be checked for new opportunities

- Lamb (1981) describes the implementation of a PO system with regular expressions and a **backwards strategy**:

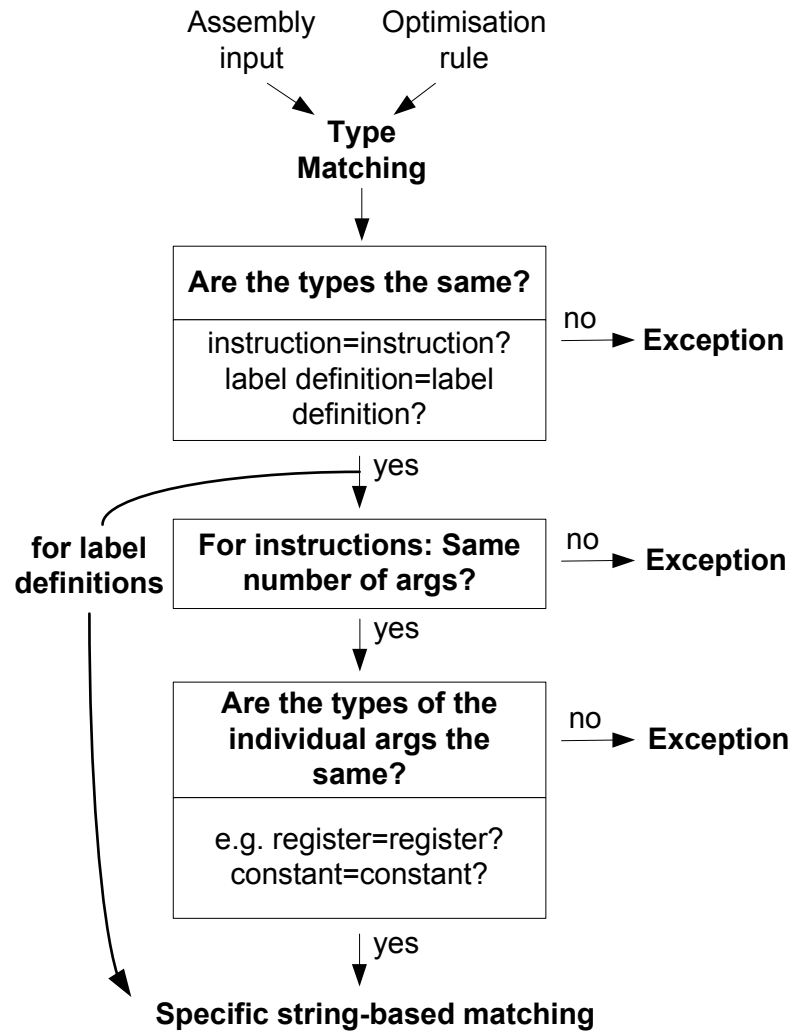


- determines how **optimisation rules are applied**: `cursor` moves forward, but rules are matched with preceding lines
- optimised lines can **immediately** be checked for new opportunities
- only **one pass** through the code is required

- Visser et al. (2004),
“Matching Objects“:
Generic pattern matching
on object level in Java



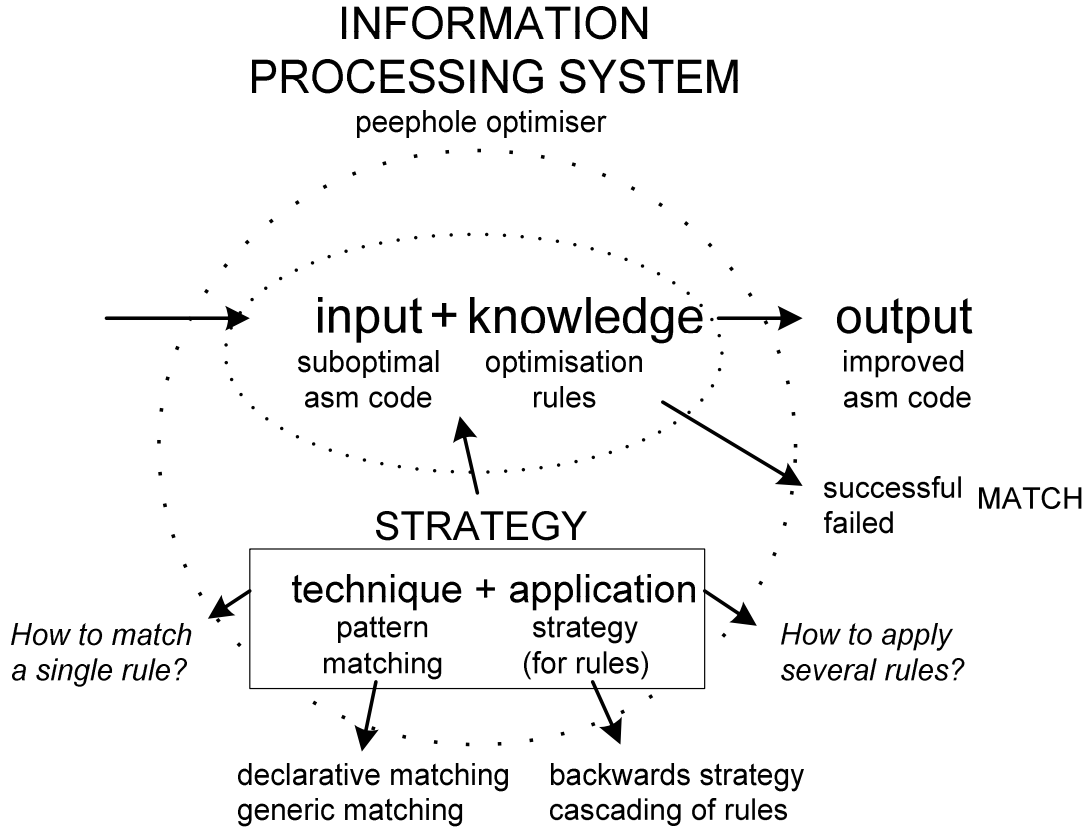
- Visser et al. (2004), “Matching Objects“: Generic pattern matching on object level in Java



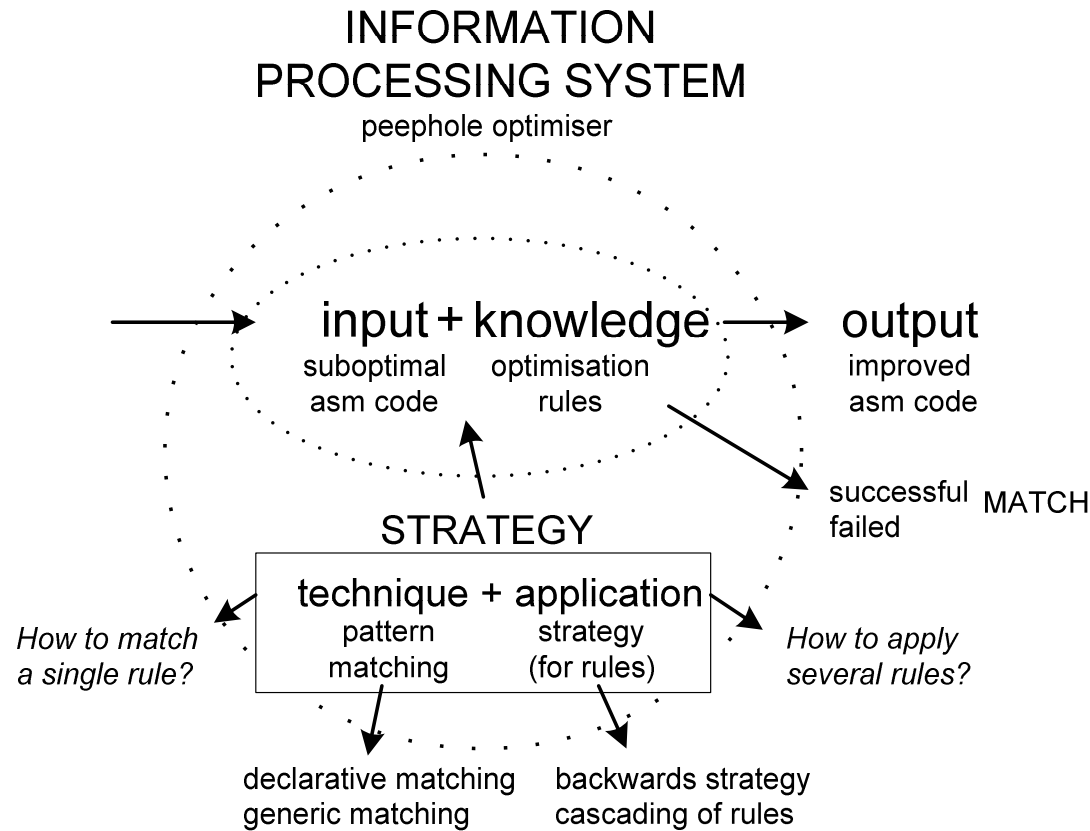
- Visser et al. (2004), “Matching Objects“: Generic pattern matching on object level in Java

→ first **type matching**, then switching to specific **string-based matching**

- What happens in a PO system?

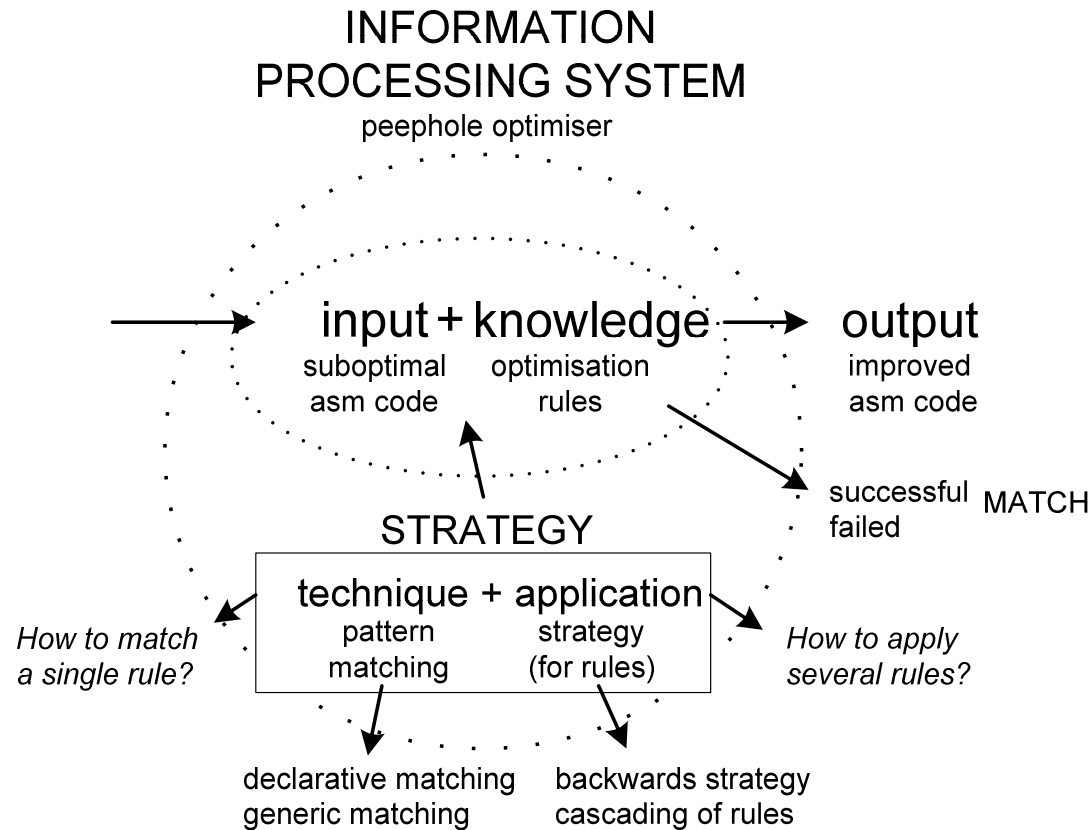


- What happens in a PO system?



• What happens in a PO system?

→ PO strategy: **pattern matching**



• What happens in a PO system?

→ **PO strategy:**
pattern matching
+
application strategy for rules

- string-based pattern matching with regular expressions as in Lamb (1981) and other classical PO systems; combined with backwards strategy

- string-based pattern matching with regular expressions as in Lamb (1981) and other classical PO systems; combined with backwards strategy

→ **declarative matching:** matching is expressed mainly with the declarative format of the optimisation rules

- string-based pattern matching with regular expressions as in Lamb (1981) and other classical PO systems; combined with backwards strategy

→ **declarative matching:** matching is expressed mainly with the declarative format of the optimisation rules

→ matching engine does not do much (code-wise)

→ meaning of code is not utilised; “dumb” technique

- string-based pattern matching with regular expressions as in Lamb (1981) and other classical PO systems; combined with backwards strategy

→ **declarative matching:** matching is expressed mainly with the declarative format of the optimisation rules

→ matching engine does not do much (code-wise)

→ meaning of code is not utilised; “dumb” technique

- uses a basic linear line-by-line lookahead method for rules like the following:

Special rules requiring a “look-ahead“ further up in the code:

```

    j&1  _$&2
    &3*  ← arbitrary number of code lines can match
    _$&4:
    jmp  _$&2
=>    jmp  _$&4
```

variable

Special rules requiring a “look-ahead“ further up in the code:

`j&1` `__$&2` ← variable
`&3*` ← arbitrary number of code lines can match
`__$&4:` ← label definition
`jmp` `__$&2`
=> `jmp` `__$&4`

Special rules requiring a “look-ahead“ further up in the code:

`j&1` `__$&2` ← variable
`&3*` ← arbitrary number of code lines can match
`__$&4:` ← label definition
`jmp` `__$&2`
=> `jmp` `__$&4`

- one has to look-ahead in the code to determine the length of the **matching part** (`&3*`)

Special rules requiring a “look-ahead“ further up in the code:

`j&1` `__$&2` ← variable
`&3*` ← arbitrary number of code lines can match
`__$&4:` ← label definition
`jmp` `__$&2`
=> `jmp` `__$&4`

- one has to look-ahead in the code to determine the length of the **matching part** (`&3*`)

→ This can be done line by line as in the declarative pattern matching strategy, or better by means of a labels table

- generic pattern matching on object level as in Visser et al. (2004) that switches to specific string-based matching if required; combined with backwards strategy

- generic pattern matching on object level as in Visser et al. (2004) that switches to specific string-based matching if required; combined with backwards strategy
- **generic matching:** handling patterns and rules generically in the matching engine

- generic pattern matching on object level as in Visser et al. (2004) that switches to specific string-based matching if required; combined with backwards strategy

→ **generic matching:** handling patterns and rules generically in the matching engine

→ uses a labels table for `look-ahead rules`; fast and clever technique

- generic pattern matching on object level as in Visser et al. (2004) that switches to specific string-based matching if required; combined with backwards strategy

→ **generic matching:** handling patterns and rules generically in the matching engine

→ uses a labels table for `look-ahead rules`; fast and clever technique

→ meaning of code is utilised + ability to do string matching

- generic pattern matching on object level as in Visser et al. (2004) that switches to specific string-based matching if required; combined with backwards strategy

→ **generic matching:** handling patterns and rules generically in the matching engine

→ uses a labels table for `look-ahead rules`; fast and clever technique

→ meaning of code is utilised + ability to do string matching

→ rules that do not match are discarded earlier during matching

- Backwards strategy prefers shorter rules over longer ones

- Backwards strategy prefers shorter rules over longer ones
- but optimisation rules often interact and disqualify each other

- Backwards strategy prefers shorter rules over longer ones
- but optimisation rules often interact and disqualify each other
- finding the best optimisation sequence is NP-complete, but finding a good combination of rules is possible

- Backwards strategy prefers shorter rules over longer ones
- but optimisation rules often interact and disqualify each other
- finding the best optimisation sequence is NP-complete, but finding a good combination of rules is possible
- **aim:** achieve optimisation of space by choosing combinations of rules that optimise most away

- Backwards strategy prefers shorter rules over longer ones
- but optimisation rules often interact and disqualify each other
- finding the best optimisation sequence is NP-complete, but finding a good combination of rules is possible
- **aim:** achieve optimisation of space by choosing combinations of rules that optimise most away
- **idea:** generic matching + backwards strategy by searching for different optimisation sequences **in parallel**, looking ahead a predetermined amount of rules

- Backwards strategy prefers shorter rules over longer ones
- but optimisation rules often interact and disqualify each other
- finding the best optimisation sequence is NP-complete, but finding a good combination of rules is possible
 - **aim:** achieve optimisation of space by choosing combinations of rules that optimise most away
- **idea:** generic matching + backwards strategy by searching for different optimisation sequences **in parallel**, looking ahead a predetermined amount of rules
 - apply the most promising sequence of rules

Performance:

- the declarative pattern matching strategy is generally slower than the generic strategy (up to 6 times)

Performance:

- the declarative pattern matching strategy is generally slower than the generic strategy (up to 6 times)
- the cascading rules strategy is slower than the generic one

Performance:

- the declarative pattern matching strategy is generally slower than the generic strategy (up to 6 times)
- the cascading rules strategy is slower than the generic one

Efficiency:

Performance:

- the declarative pattern matching strategy is generally slower than the generic strategy (up to 6 times)
- the cascading rules strategy is slower than the generic one

Efficiency:

- the cascading rules strategy optimises best with specific look-ahead settings for cases where a combination of rules can be applied

Performance:

- the declarative pattern matching strategy is generally slower than the generic strategy (up to 6 times)
- the cascading rules strategy is slower than the generic one

Efficiency:

- the cascading rules strategy optimises best with specific look-ahead settings for cases where a combination of rules can be applied
- the generic strategies perform better than the declarative strategy regarding look-ahead rules (→ labels table)

- **research question:** Can PO be achieved even more successfully by utilising the meaning of the code? How?

- **research question:** Can PO be achieved even more successfully by utilising the meaning of the code? How?

→ **PO:** pattern matching strategy + rule application strategy

- **research question:** Can PO be achieved even more successfully by utilising the meaning of the code? How?

- **PO:** pattern matching strategy + rule application strategy

- implemented each two examples for these: declarative, generic and backwards, cascading rules

- **research question:** Can PO be achieved even more successfully by utilising the meaning of the code? How?

- **PO:** pattern matching strategy + rule application strategy

- implemented each two examples for these: declarative, generic and backwards, cascading rules

- **results:** better alternative to matching by regular expressions is a **generic pattern matching strategy** that uses a **labels table** and **backwards matching + cascading of rules** (for assembly code for which rules interact much)

- **research question:** Can PO be achieved even more successfully by utilising the meaning of the code? How?

- **PO:** pattern matching strategy + rule application strategy

- implemented each two examples for these: declarative, generic and backwards, cascading rules

- **results:** better alternative to matching by regular expressions is a **generic pattern matching strategy** that uses a **labels table** and **backwards matching + cascading of rules** (for assembly code for which rules interact much)

- cascading of rules makes editing the rules file unnecessary and achieves optimisation of space

- Labels table: Currently 2 cases of rules are handled; Extensions?

- Labels table: Currently 2 cases of rules are handled; Extensions?
- Generic matching: employed at level 1; Deeper?

- Labels table: Currently 2 cases of rules are handled; Extensions?
 - Generic matching: employed at level 1; Deeper?
 - Cascading rules strategy: Improvements? Automatically finding the best look-ahead (machine learning etc.) ?
- Analysis of look-ahead settings with different input files and rules sets

- Labels table: Currently 2 cases of rules are handled; Extensions?
- Generic matching: employed at level 1; Deeper?
- Cascading rules strategy: Improvements? Automatically finding the best look-ahead (machine learning etc.) ?
→ Analysis of look-ahead settings with different input files and rules sets
- More intelligent strategies? AI techniques?

Questions?

Thank you!

