this::=sle'16

Efficient Development of Consistent Projectional Editors using Grammar Cells

Markus Völter, Tamás Szabó, Sascha Lisson,
Bernd Kolb, Sebastian Erdweg, Thorsten Berger

voelter {ingenieurbüro für softwaretechnologie // itemis

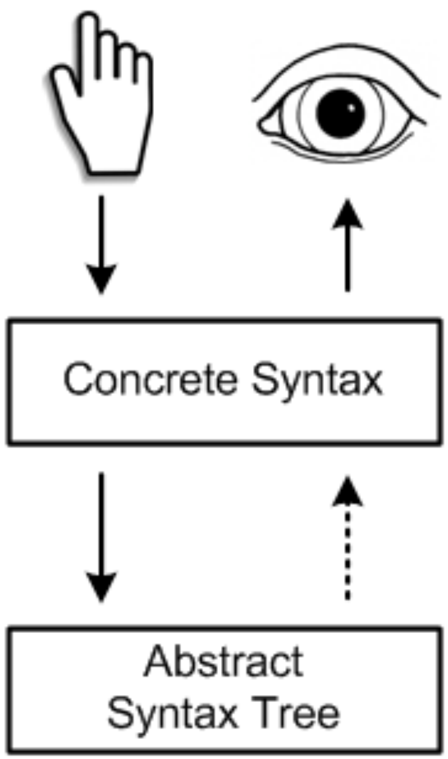
voelter@acm.org
www.voelter.de
@markusvoelter

Grammar Cells 1

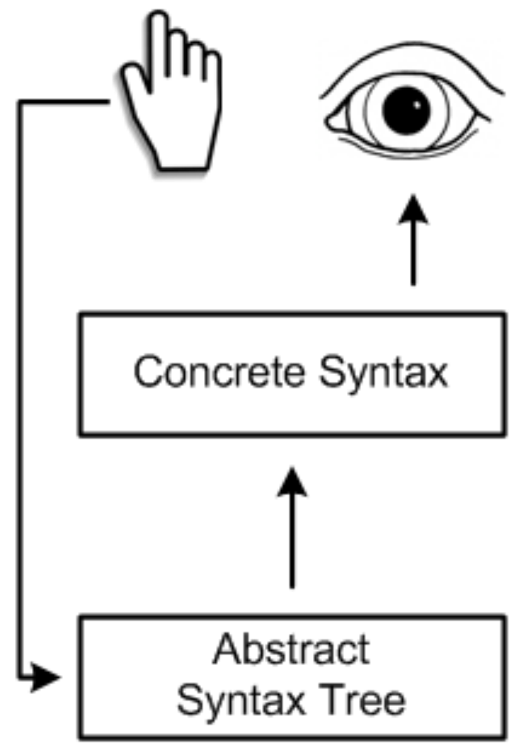
Why Projectional Editors

[Projectional Editing]

Parsing



Projectional Editing



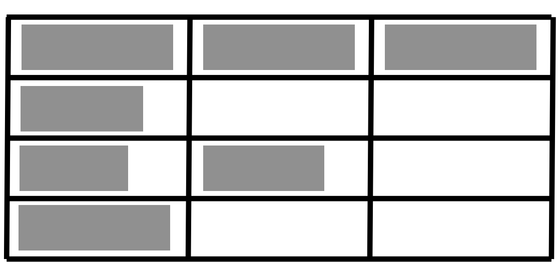
[Projectional Editing]

Syntactic Flexibility

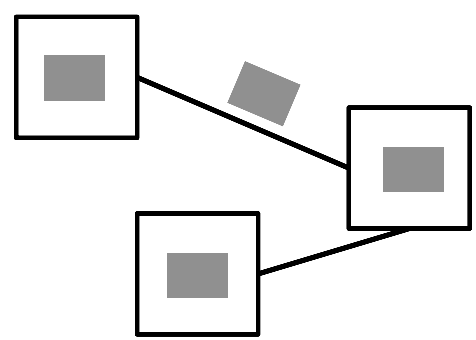
Regular Code/Text

Mathematical

Tables



Graphical



[Projectional Editing]
Syntactic Flexibility

Regular Code/Text

```
//[ A documentation comment with references ]
  [ to @arg(data) and @arg(dataLen)         ]
void aSummingFunction(int8[] data, int8 dataLen) {
  int16 sum;
  for (int8 i = 0; i < dataLen; i++) {
    sum += data[i];
  } for
} aSummingFunction (function)
```

Mathematical

```
double midnight2(int32 a, int32 b, int32 c) {
```

$$\text{return } \frac{-b + \sqrt{b^2 - \sum_{i=1}^{4} a * c}}{2 * a};$$

```
} midnight2 (function)
```

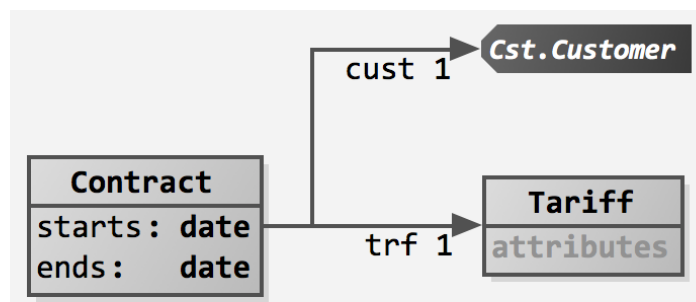
Tables

```
int16 decide(int8 spd, int8 alt) {
  return
```

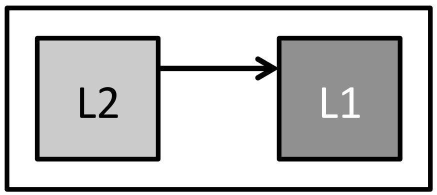
	spd > 0	spd > 100
alt < 0	1	1
alt == 0	10	20
alt > 0	30	40
alt > 100	50	60

```
  otherwise 0;
} decide (function)
```

Graphical



[Projectional Editing]
Language Composition



Separate Files

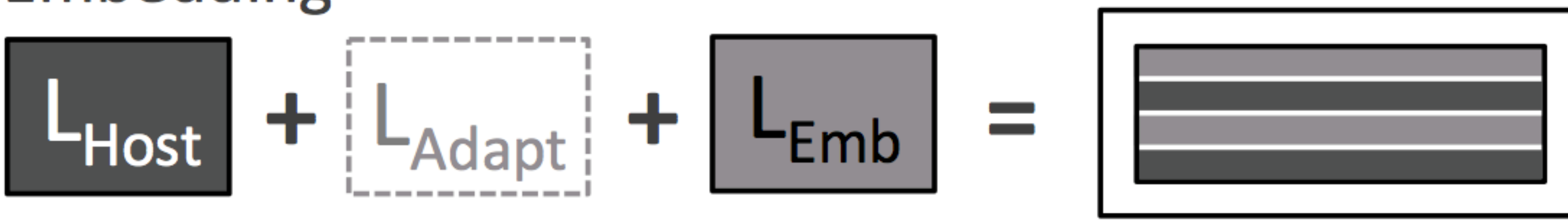
Type System
Transformation
Constraints

In One File

Type System
Transformation
Constraints
Syntax
IDE

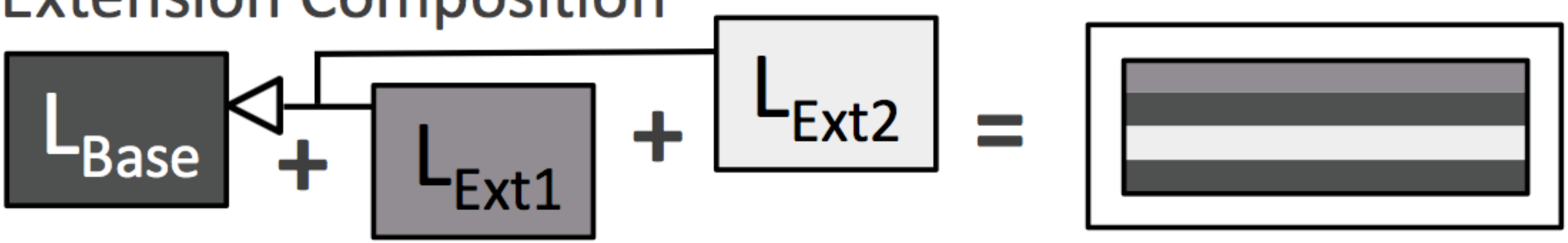
[Projectional Editing]
Language Composition

Embedding



Extension

Extension Composition



Grammar Cells | 2

The Usability Issue

[Projectional Editing]
Study Results on Editor Usability

People **prefer MPS** over conventional IDEs

MPS more is **more efficient** than normal IDEs

MPS more is **more productive** than normal IDEs

MPS makes it easier to create **correct programs**

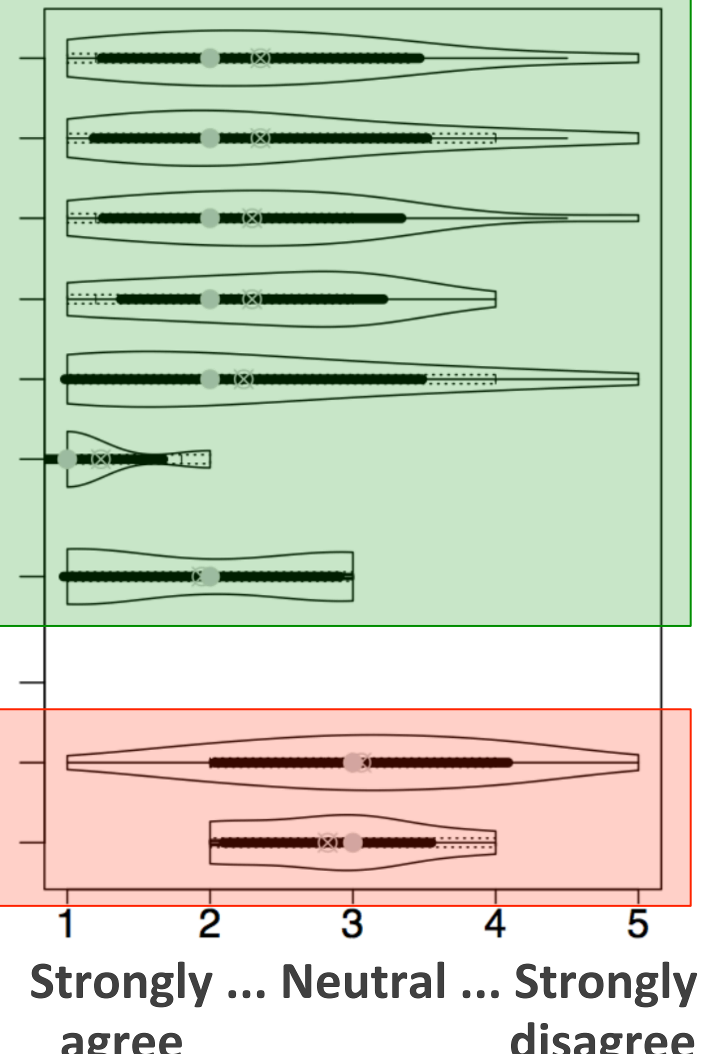
MPS enforces a **structurally correct AST**

People benefit from **language modularity**

People benefit from the **flexible notations**

The experience with **learning MPS** is mixed.

It takes some time to **get used to MPS**



1980 1990 2000 2010

Early Days

The tree dominated the editing experience.

Enter new nodes based on tree structure.

Select and modify based on tree structure.

Modify through menu-based user interactions.

No user acceptance because too slow, and not like text editing for textual notations.

1980 1990 2000 2010

Resurgence

Textual Notations can be edited „linearly“.

Based on little tree-transformations triggered by editing actions.

Those actions had to be built manually.

Effort for good editors is high.

Int{er|ra}-Language Consistency is a problem.

User acceptance was mostly there, but few good editors ever built.

1980 1990 2000 2010

GC

Textual Notations can be edited „linearly“.
Based on little tree-transformations triggered by editing actions.
Actions automatically derived from higher-level semantically rich editor descriptions.

Effort for building good editors has gone to almost zero. Editors are consistent.

1980 1990 2000 2010

GC

Grammar Cells

Enter new nodes based on tree structure.

Enter nodes mostly linearly/textually.

Modify through menu-based user interactions

Modify mostly through typing, deleting, etc.

Effort for good editors is high.

Reduced through abstraction & code generation.

Int{er|ra}-Language Consistency is a problem.

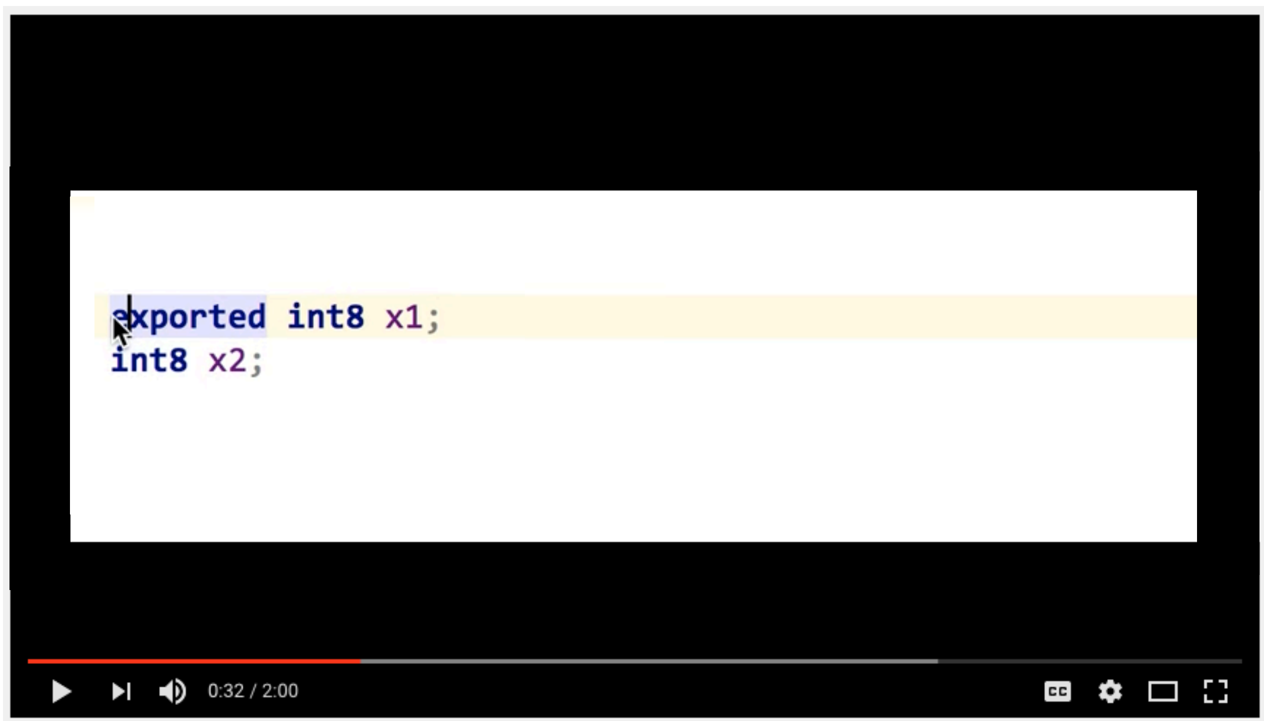
Consistency is there b/c of idiomatics.

Select and modify based on tree structure.

(This issue is still there, unchanged.)

Grammar Cells Demo

https://www.youtube.com/watch?v=QxXHtp90Fcs

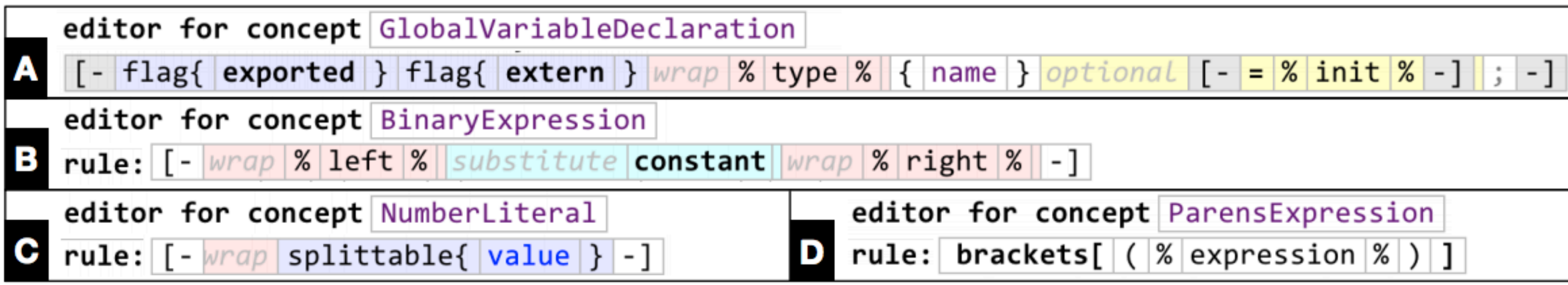


Grammar Cells

4

How Grammar Cells Work

More semantics
in the editor definitions



based on problems identified in user studies and accumulated experience from dozens of developers and languages.

Declarative Descriptions

for the most typical editor actions

flag C, $C.cld : Boolean$ **in** [flag[l^child[C.cld]]]

optional C, $C.cld : T$
in [optional[list[l^constant[t], child[C.cld]]]]

wrap C, $C.cld : T$ **in** [wrap[child[C.cld]]]

substitute C_1 **in** [substitute[l^const]]

brackets C, P, $P.cld : D$, $C <: D$
in [brackets[l^constant[open],
child[C.cld], constant[close]^r]]

Key for the notation:
$C, C_1, C_2, D, P, T \in \mathbb{C}$ (*language concepts*) **in** [editor] $\Longrightarrow$ **action**(*params* | *typed text* $\mapsto$ `executed code`)

Declarative Descriptions
for the most typical editor actions

translated to the available action primitives in PE

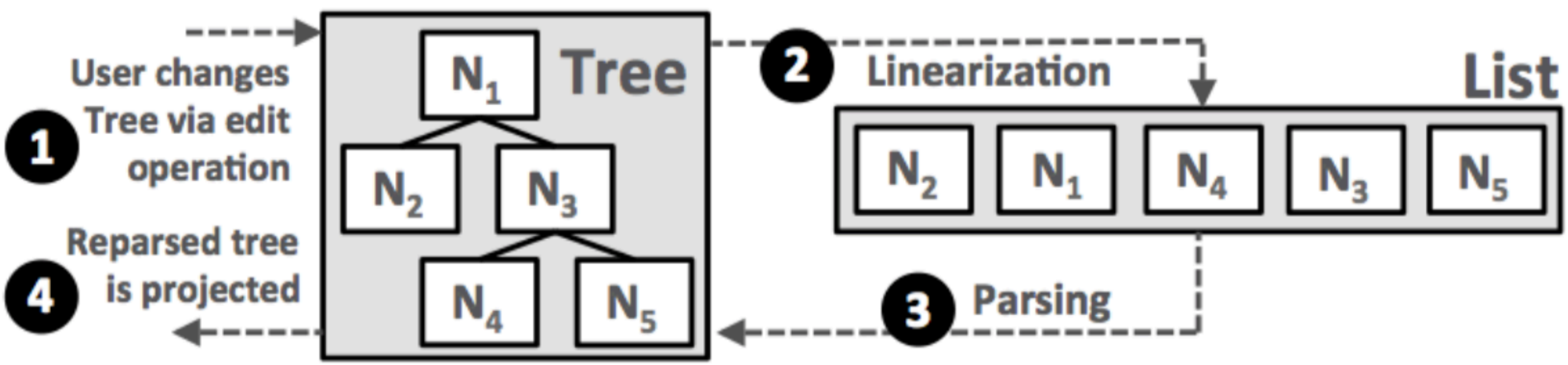
flag	$C,\ C.cld : Boolean$ **in** [flag[l^child[C.cld]]]	$\Longrightarrow$	$\mathbf{side}(c@l : C \mid \texttt{nameOfLink}(C.cld) \mapsto c.cld = true)$ $\mathbf{delete}(c@l : C \mid c.cld = false)$
optional	$C,\ C.cld : T$ **in** [optional[list[l^constant[t], child[C.cld]]]]	$\Longrightarrow$	$\mathbf{side}(c@l : C \mid t \mapsto c.cld = \texttt{new}\ T)$ $\mathbf{delete}(c@l : C \mid \texttt{delete}(c.cld))$
wrap	$C,\ C.cld : T$ **in** [wrap[child[C.cld]]]	$\Longrightarrow$	$\mathbf{subst}(\mid t : T \mapsto c = \texttt{new}\ C,\ c.cld = t,\ \texttt{replace}(t \leftarrow c))$
substitute	C_1 **in** [substitute[l^const]]	$\Longrightarrow$	$\forall C_2 \in \texttt{structuralMatches}(C_1) :$ $\mathbf{subst}(c_1@l : C_1 \mid C_m.const \mapsto c_2 = \texttt{new}\ C_2,$ $\texttt{copyStructure}(c_2 \leftarrow c_1),\ \texttt{replace}(c_1 \leftarrow c_2))$
brackets	$C,\ P,\ P.cld : D,\ C <: D$ **in** [brackets[l^constant[open], child[C.cld], constant[close]^r]]	$\Longrightarrow$	$\mathbf{side}(c@l : C \mid open \mapsto t : D = \texttt{reparse}(c),\ \texttt{replace}(c \leftarrow t))$ $\mathbf{side}(c@r : C \mid close \mapsto t : D = \texttt{reparse}(c),\ \texttt{replace}(c \leftarrow t))$ $\mathbf{delete}(c@l : C \mid t : D = \texttt{reparse}(c),\ \texttt{replace}(c \leftarrow t))$ $\mathbf{delete}(c@r : C \mid t : D = \texttt{reparse}(c),\ \texttt{replace}(c \leftarrow t))$

Key for the notation:

$C, C_1, C_2, D, P, T \in \mathbb{C}$ (*language concepts*) **in** [editor] $\Longrightarrow$ $\mathbf{action}(params \mid typed\ text \mapsto \texttt{executed code})$

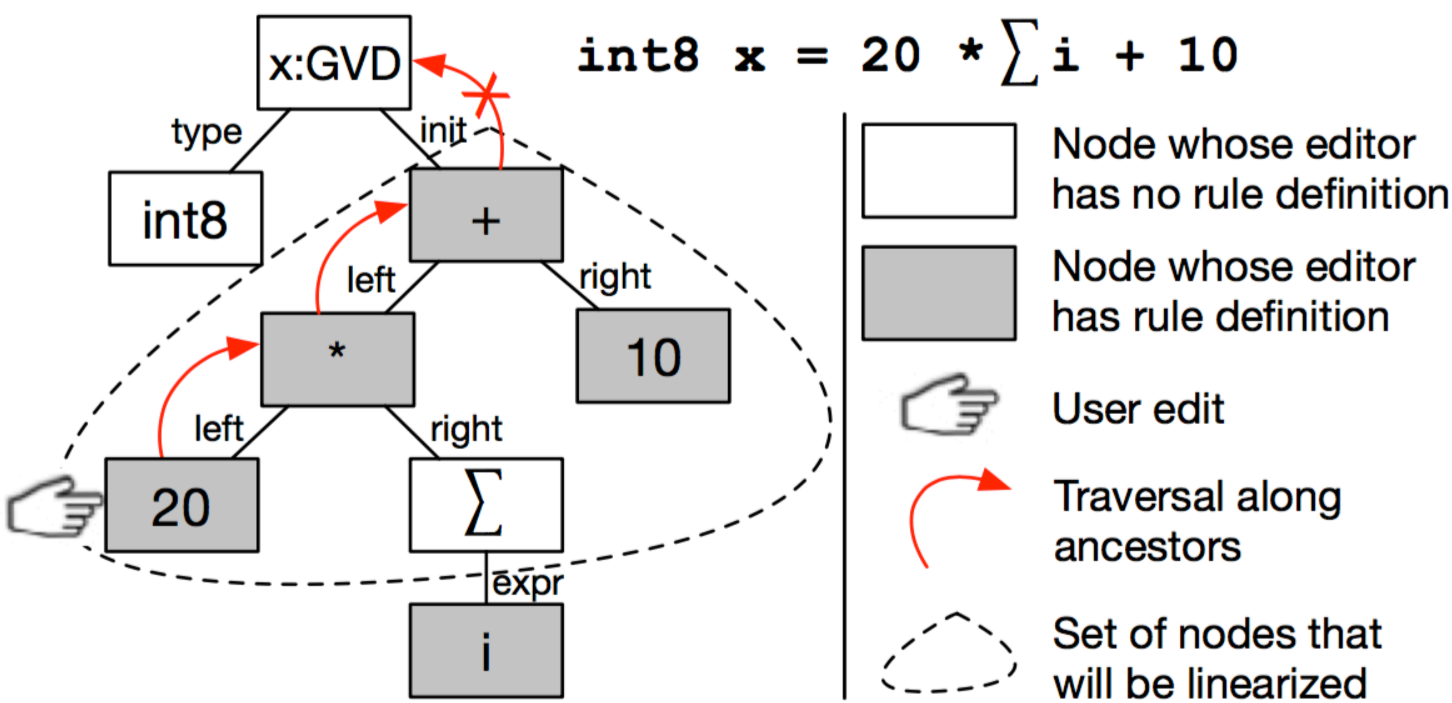
Integrated Parsing
for expressions

to deal with precedence, associativity and cross-tree editing.



Integrated Parsing
for expressions

complex, non-text tokens remain intact. Notation mixing still possible.



Grammar
Cells

5

Wrap
Up

PEs have many advantages.

Mixing Notations, Language Composition.

Editing Experience was a Challenge.

Textual Notations were not editable as in text editors.

Editor behavior must be consistent

within one and across several (composed) languages

Grammar Cells support „nice" editors

with very limited editor development effort.

Editor End-User
Language Dev } **feedback is very positive**

„this changes the game for Projectional Editors"

PEs have many advantages.

Mixing Notations, Language Composition.

Grammar Cells make exploiting these benefits a real option!

Editor End-User
Language Dev
} feedback is very positive

„this changes the game for Projectional Editors“